Introduction

Intro text for the entire document, outline structure and define the purpose

1 xAPI Data Retrival

The following section describes how to Query an LRS to retrieve a statement or a set of statements [1]

```
Agent = "agent={"account":
                    {"homePage": "https://example.homepage",
                     "name": 123456}}"

Since = "since=2018-07-20T12:08:47Z"

Until = "until=2018-07-21T12:08:47Z"

Base = "https://example.endpoint/statements?"

endpoint = Base + Agent + "&" + Since + "&" + Until

Auth = Hash generated from basic auth

S = curl -X GET -H "Authorization: Auth"
           -H "Content-Type: application/json"
           -H "X-Experience-API-Version: 1.0.3"
           Endpoint
```

- Update as needed to reflect the needs of an LRS query

2 xAPI Z Specifications

An xAPI statement(s) is only defined abstractly within the context of Z. A concrete definition for an xAPI statement(s) it outside the scope of this specification.

2.1 Basic Types

$IFI ::= mbox \mid mbox\_sha1sum \mid openid \mid account$

- Type unique to Agents and Groups, The concrete definition of the listed values is outside the scope of this specification

[1] S is the set of all statements parsed from the statements array within the HTTP response to the Curl request. It may be possible that multiple Curl requests are needed to retrieve all query results. If multiple requests are necessary, S is the result of concatenating the result of each request into a single set

$OBJECTTYPE ::= Agent \mid Group \mid SubStatement \mid StatementRef \mid Activity$

- A type which can be present in all activities as defined by the xAPI specification

$INTERACTIONTYPE ::= true{-}false \mid choice \mid fill{-}in \mid long{-}fill{-}in \mid matching \mid performance \mid sequencing \mid likert \mid numeric \mid other$

- A type which represents the possible interactionTypes as defined within the xAPI specification

$INTERACTIONCOMPONENT ::= choices \mid scale \mid source \mid target \mid steps$

- A type which represents the possible interaction components as defined within the xAPI specification
- the concrete definition of the listed values is outside the scope of this specification

$CONTEXTTYPES ::= parent \mid grouping \mid category \mid other$

- A type which represents the possible context types as defined within the xAPI specification

$[STATEMENT]$

- Basic types for the results of querying an LRS

$[AGENT, GROUP]$

- Basic types for Agents and collections of Agents

2.2 Schema

Z schema for statements and the components of statements

2.2.1 Id Schema

$$
\begin{array}{|l}
\hline
\textit{Id} \\
\hline
id : \mathbb{F}_1 \#1 \\
\hline
\end{array}
$$

- the schema Id introduces the component id which is a non-empty finite set of 1 value

2.2.2 Schemas for Agents and Groups

$$
\begin{array}{|l}
\hline
\textit{Agent} \\
\hline
agent : AGENT \\
objectType : OBJECTTYPE \\
name : \mathbb{F}_1 \#1 \\
ifi : IFI \\
\hline
objectType = Agent \\
agent = \{ifi\} \cup \mathbb{P}\{name, objectType\} \\
\hline
\end{array}
$$

- The schema $Agent$ introduces the component $agent$ which is a set consisting of an ifi and optionally an $objectType$ and/or $name$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Member\ }} \\
Agent \\
member : \mathbb{F}_1 \\
\hline
member = \{a : AGENT \mid \forall a : a_0..a_n \bullet a = agent\}
\end{array}
$$

- The schema $Member$ introduces the component $member$ which is a set of objects a, where for every a within $a_0..a_n$, a is an $agent$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Group\ }} \\
Member \\
group : GROUP \\
objectType : OBJECTTYPE \\
ifi : IFI \\
\hline
name : \mathbb{F}_1 \#1 \\
objectType = Group \\
group = \{objectType, name, member\} \vee \{objectType, member\} \vee \\
\qquad\qquad \{objectType, ifi\} \cup \mathbb{P}\{name, member\}
\end{array}
$$

- The schema $Group$ introduces the component $group$ which is of type $GROUP$ and is a set of either $objectType$ and $member$ with optionaly $name$ or $objectType$ and ifi with optionally $name$ and/or $member$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Actor\ }} \\
Agent \\
Group \\
actor : AGENT \vee GROUP \\
\hline
actor = agent \vee group
\end{array}
$$

- The schema $Actor$ introduces the component $actor$ which is either an $agent$ or $group$

2.2.3 Verb Schema

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Verb\ }} \\
Id \\
display, verb : \mathbb{F}_1 \\
\hline
verb = \{id, display\} \vee \{id\}
\end{array}
$$

- The schema $Verb$ introduces the component $verb$ which is a set that consists of either id and the finite set $display$ or just id

2.2.4 Object Schema

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Extensions}} \\
\hline
extensions, extensionVal : \mathbb{F}_1 \\
extensionId : \mathbb{F}_1 \ \#1 \\
\hline
extensions = \{e : (extensionId, extensionVal) \ | \forall i, j : e_i..e_j \bullet \\
\qquad (extensionId_i, extensionVal_i) \vee (extensionId_i, extensionVal_j) \wedge \\
\qquad (extensionId_j, extensionVal_i) \vee (extensionId_j, extensionVal_j) \wedge \\
\qquad extensionId_i \neq extensionId_j\} \\
\hline
\end{array}
$$

- The schema *Extensions* introduces the component *extensions* which is a non-empty finite set that consists of ordered pairs of *extensionId* and *extensionVal*. Different *extensionId*s can have the same *extensionVal* but there can not be two identical *extensionId* values
- *extensionId* is a non-empty finite set with one value
- *extensionVal* is a non-empty finite set

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{InteractionActivity}} \\
\hline
interactionType : INTERACTIONTYPE \\
correctResponsePattern : \mathrm{seq}_1 \\
interactionComponent : INTERACTIONCOMPONENT \\
\hline
interactionActivity = \{interactionType, correctReponsePattern, interactionComponent\} \vee \\
\qquad \{interactionType, correctResponsePattern\} \\
\hline
\end{array}
$$

- The schema *InteractionActivity* introduces the component *interactionActivity* which is a set of either *interactionType* and *correctResponsePattern* or *interactionType* and *correctResponsePattern* and *interactionComponent*

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Definition}} \\
\hline
InteractionActivity \\
Extensions \\
definition, name, description : \mathbb{F}_1 \\
type, moreInfo : \mathbb{F}_1 \ \#1 \\
\hline
definition = \mathbb{P}_1\{name, description, type, moreInfo, extensions, interactionActivity\} \\
\hline
\end{array}
$$

- The schema *Definition* introduces the component *definition* which is the non-empty, finite power set of *name*, *description*, *type*, *moreInfo* and *extensions*

$$
\begin{array}{|l}
\hline
\textit{Object} \\
Id \\
Definition \\
Agent \\
Group \\
Statement \\
objectTypeA, objectTypeS, objectTypeSub, objectType : OBJECTTYPE \\
substatement : STATEMENT \\
object : \mathbb{F}_1 \\
\hline
substatement = statement \\
objectTypeA = Activity \\
objectTypeS = StatementRef \\
objectTypeSub = SubStatement \\
objectType = objectTypeA \vee objectTypeS \\
object = \{id\} \vee \{id, objectType\} \vee \{id, objectTypeA, definition\} \\
\qquad \vee \{id, definition\} \vee \{agent\} \vee \{group\} \vee \{objectTypeSub, substatement\} \\
\qquad \vee \{id, objectTypeA\} \\
\hline
\end{array}
$$

- The schema *Object* introduces the component *object* which is a non-empty finite set of either *id*, *id* and *objectType*, *id* and *objectTypeA* and *definition*, *agent*, *group*, or *substatement*
- The schema *Statement* and the corresponding component *statement* will be defined later on in this specification

2.2.5 Result Schema

$$
\begin{array}{|l}
\hline
\textit{Score} \\
score : \mathbb{F}_1 \\
scaled, min, max, raw : \mathbb{Z} \\
\hline
scaled = \{n : \mathbb{Z} \mid -1.0 \leq n \leq 1.0\} \\
min = n < max \\
max = n > min \\
raw = raw = \{n : \mathbb{Z} \mid min \leq n \leq max\} \\
score = \mathbb{P}_1\{scaled, raw, min, max\} \\
\hline
\end{array}
$$

- The schema *Score* introduces the component *score* which is the non-empty powerset of *min*, *max*, *raw* and *scaled*

$$
\begin{array}{|l}
\hline
\textit{Result} \\
\textit{Score} \\
\textit{Extensions} \\
success, completion, response, duration : \mathbb{F}_1 \#1 \\
result : \mathbb{F}_1 \\
\hline
success = true \lor false \\
completion = true \lor false \\
result = \mathbb{P}_1\{score, success, completion, response, duration, extensions\} \\
\hline
\end{array}
$$

- The schema *Result* introduces the component *result* which is the non-empty power set of *score*, *success*, *completion*, *response*, *duration* and *extensions*

2.2.6 Context Schema

$$
\begin{array}{|l}
\hline
\textit{Instructor} \\
\textit{Agent} \\
\textit{Group} \\
instructor : AGENT \lor GROUP \\
\hline
instructor = agent \lor group \\
\hline
\end{array}
$$

- The schema *Instructor* introduces the component *instructor* which can be ether an *agent* or a *group*

$$
\begin{array}{|l}
\hline
\textit{Team} \\
\textit{Group} \\
team : GROUP \\
\hline
team = group \\
\hline
\end{array}
$$

- The schema *Team* introduces the component *team* which is a *group*

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\textit{Context}\hspace{20em}}} \\
Instructor \\
Team \\
Object \\
Extensions \\
registration, revision, platform, language : \mathbb{F}_1 \#1 \\
parentT, groupingT, categoryT, otherT : CONTEXTTYPES \\
contextActivities, statement : \mathbb{F}_1 \\
\hline
statement = object \setminus (id, objectType, agent, group, definition) \\
parentT = parent \\
groupingT = grouping \\
categoryT = category \\
otherT = other \\
contextActivity = \{ca : object \setminus (agent, group, objectType, objectTypeSub, substatement)\} \\
contextActivityParent = (parentT, contextActivity) \\
contextActivityCategory = (categoryT, contextActivity) \\
contextActivityGrouping = (groupingT, contextActivity) \\
contextActivityOther = (otherT, contextActivity) \\
contextActivities = \mathbb{P}_1\{contextActivityParent, contextActivityCategory, \\
\qquad\qquad contextActivityGrouping, contextActivityOther\} \\
context = \mathbb{P}_1\{registration, instructor, team, contextActivities, revision, \\
\qquad\qquad platform, language, statement, extensions\} \\
\hline
\end{array}
$$

- The schema *Context* introduces the component *context* which is the non-empty powerset of *registration*, *instructor*, *team*, *contextActivities*, *revision*, *platform*, *language*, *statement* and *extensions*

2.2.7 Timestamp and Stored Schema

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\textit{Timestamp}\hspace{20em}}} \\
timestamp : \mathbb{F}_1 \#1 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\textit{Stored}\hspace{20em}}} \\
stored : \mathbb{F}_1 \#1 \\
\hline
\end{array}
$$

- The schema *Timestamp* and *stored* introduce the components *timestamp* and *stored* respectively. Each are non-empty finite sets containing one value

2.2.8 Attachements Schema

$$
\begin{array}{|l}
\underline{\quad Attachments \quad} \\
display, description, attachment, attachments : \mathbb{F}_1 \\
usageType, sha2, fileUrl, contexntType : \mathbb{F}_1 \#1 \\
length : \mathbb{N} \\
\hline
attachment = \{usageType, display, contentType, length, sha2\} \cup \mathbb{P}\{description, fileUrl\} \\
attachments = \{a : attachment\} \\
\hline
\end{array}
$$

2.2.9 Statement and Statements Schema

$$
\begin{array}{|l}
\underline{\quad Statement \quad} \\
Id \\
Actor \\
Verb \\
Object \\
Result \\
Context \\
Timestamp \\
Stored \\
Attachements \\
statement, \$: STATEMENT \\
\hline
statement = \{actor, verb, object, stored\} \cup \\
\qquad\qquad \mathbb{P}\{\text{id}, result, context, timestamp, attachments\} \\
\$ \leadsto statement \\
\hline
\end{array}
$$

- The schema *Statement* introduces the component *statement* which consists of the components *actor*, *verb*, *object* and *stored* and the optional components *id*, *result*, *context*, *timestamp*, and/or *attachments*
- The schema *Statement* also binds the component *statement* to the variable $ so that JSONPath can be used within Operation schemas which require reaching into a *statement*. This is accomplished by using the . (select) notation starting at $ (root) and navigating into subsequent components of the *statement*

$$
\begin{array}{|l}
\underline{\quad Statements \quad} \\
Statement \\
statements : \mathbb{F}_1 \\
\hline
statements = \{s : statement\} \\
\hline
\end{array}
$$

- The schema *Statements* introduces the component *statements* which is a non-empty finite set of components *statement*

3 Question 1 Name

intro text for the question

3.1 Statements

3.1.1 Ideal Statements

paragraph or list describing the ideal input statements

3.1.2 statement parameters to utilize

- first param
- second param
- third param

3.1.3 TLA Statement problems

paragraph talking about known data issues within current TLA implementation

3.2 Algorithm

3.2.1 Summary

1. step 1
2. step 2
3. step 3

3.3 Z Specification

3.3.1 Introduce Basic Types

Template [Name of variable(s) of type set]

Example [X]

3.3.2 Example Schema

Basic unit of specification, defines state variables, system state, operations, etc.

Template

$SchemaName$

$VariableDeclarations$

$Predicate/Invariants$

Example

$$
\begin{array}{|l}
\hline
\textit{Counter} \\
ctx : \mathbb{N} \\
\hline
0 \leq ctr \leq max \\
\hline
\end{array}
$$

Variables

$$
\begin{array}{|l}
\hline
\textit{Counter} \\
ctx : \mathbb{N} \\
\hline
\\
\hline
\end{array}
$$

- the variable ctx is a natural number

Predicates

$$
\begin{array}{|l}
\hline
\textit{Counter} \\
\hline
0 \leq ctr \leq max \\
\hline
\end{array}
$$

- ctr is greater than or equal to 0
- ctr is less than or equal to max

3.3.3 Initialisation

The starting conditions

Template

$$
\begin{array}{|l}
\hline
Init[VarName] \\
NameOfExistingSchema \\
\hline
InitStateOfVarsWithinRefSchema \\
\hline
\end{array}
$$

Example

$$
\begin{array}{|l}
\hline
InitCounter \\
Counter \\
\hline
ctr = 0 \\
\hline
\end{array}
$$

- the value of the counter starts at 0

3.3.4 Operations

an operation is specified in Z with a predicate relating the state before and after the invocation of that operation

Template

$$
\begin{array}{|l}
\hline
\textit{OperationName} \\
\Delta \textit{SchemaName} \\
\textit{inputParam}? : \textit{SomeType} \\
\textit{outputParam}! : \textit{SomeType} \\
\hline
\textit{InvariantPredicate} \\
\textit{NewValForVar}' = \textit{OperationOnInput/OutputParams} \\
\hline
\end{array}
$$

Example

$$
\begin{array}{|l}
\hline
\textit{Increment} \\
\Delta \textit{Counter} \\
\hline
\textit{ctr} < \textit{max} \\
\textit{ctr}' = \textit{crt} + 1 \\
\hline
\end{array}
$$

- There is an implicit conjunction (logical-and) between successive lines of the predicate

$$
\begin{array}{|l}
\hline
\textit{Decrement} \\
\Delta \textit{Counter} \\
d? : \mathbb{N} \\
\hline
\textit{ctr} \geq d? \\
\textit{ctr}' = \textit{ctr} - d? \\
\hline
\end{array}
$$

- input params suffixed with ?

$$
\begin{array}{|l}
\hline
\textit{Display} \\
\Xi \textit{Counter} \\
c! : \mathbb{N} \\
\hline
c! = \textit{ctr} \\
\hline
\end{array}
$$

- output params suffixed with !
- the greek symbol means that the operation cannot change the state of Counter

3.4 Pseudocode

Algorithm 1: How to write algorithms

```
Input: this text
Result: how to write algorithm with LaTeX2e
initialization;
while not at end of this document do
    read current;
    if understand then
        go to next section;
        current section becomes this one;
    else
        go back to the beginning of current section;
    end
end
```

3.5 Result JSON Schema

JSON schema describing the returned data structure

3.6 Visualization Description

description of the associated visualization in english

3.7 Visualization prototype

This section will be updated to a prototype viz

4 Question 2 Name

intro text for the question

4.1 Statements

4.1.1 Ideal Statements

paragraph or list describing the ideal input statements

4.1.2 statement parameters to utilize

- first param
- second param
- third param

4.1.3 TLA Statement problems

paragraph talking about known data issues within current TLA implementation

4.2 Algorithm

4.2.1 Summary

1. step 1
2. step 2
3. step 3

4.3 Z Specification

4.3.1 Introduce Basic Types

Template [Name of variable(s) of type set]

Example [X]

4.3.2 Example Schema

Basic unit of specification, defines state variables, system state, operations, etc.

Template

$SchemaName$

$VariableDeclarations$

$Predicate/Invariants$

Example

$Counter$

$ctx : \mathbb{N}$

$0 \leq ctr \leq max$

Variables

$Counter$

$ctx : \mathbb{N}$

- the variable ctx is a natural number

Predicates

$$\begin{array}{|l}\hline \textit{Counter} \\ \hline 0 \leq ctr \leq max \\ \hline\end{array}$$

- ctr is greater than or equal to 0
- ctr is less than or equal to max

4.3.3 Initialisation

The starting conditions

Template

$$\begin{array}{|l}\hline Init[VarName] \\ NameOfExistingSchema \\ \hline InitStateOfVarsWithinRefSchema \\ \hline\end{array}$$

Example

$$\begin{array}{|l}\hline InitCounter \\ Counter \\ \hline ctr = 0 \\ \hline\end{array}$$

- the value of the counter starts at 0

4.3.4 Operations

an operation is specified in Z with a predicate relating the state before and after the invocation of that operation

Template

$$\begin{array}{|l}\hline OperationName \\ \Delta SchemaName \\ inputParam? : SomeType \\ outputParam! : SomeType \\ \hline InvariantPredicate \\ NewValForVar' = OperationOnInput/OutputParams \\ \hline\end{array}$$

Example

$$\begin{array}{|l}
\hline
\textit{Increment} \\
\Delta Counter \\
\hline
ctr < max \\
ctr' = crt + 1 \\
\hline
\end{array}$$

- There is an implicit conjunction (logical-and) between successive lines of the predicate

$$\begin{array}{|l}
\hline
\textit{Decrement} \\
\Delta Counter \\
d? : \mathbb{N} \\
\hline
ctr \geq d? \\
ctr' = ctr - d? \\
\hline
\end{array}$$

- input params suffixed with ?

$$\begin{array}{|l}
\hline
\textit{Display} \\
\Xi Counter \\
c! : \mathbb{N} \\
\hline
c! = ctr \\
\hline
\end{array}$$

- output params suffixed with !
- the greek symbol means that the operation cannot change the state of Counter

4.4 Pseudocode

Algorithm 2: How to write algorithms

```
Input: this text
Result: how to write algorithm with LaTeX2e
initialization;
while not at end of this document do
    read current;
    if understand then
        go to next section;
        current section becomes this one;
    else
        go back to the beginning of current section;
    end
end
```

4.5 Result JSON Schema

JSON schema describing the returned data structure

4.6 Visualization Description

description of the associated visualization in english

4.7 Visualization prototype

This section will be updated to a prototype viz