1 Timeline Of Learner Success

As learners engage in activities supported by a learning ecosystem, they will build up a history of learning experiences. When the digital resources of that learning ecosystem adhere to a framework dedicated to supporting and understanding the learner, such as the Total Learning Architecture (TLA), it becomes possible to retell their learning story through data and data visualization. One important aspect of that story is the' learners history of success.

1.1 Ideal Statements

In order to accurately portray a learner's timeline of success, there are a few base requirements of the data produced by a Learning Record Provider (LRP). They are as follows:

- the learner must be uniquely and consistently identified across all statements
- learning activities which evaluate a learner's understanding of material must report if the learner was successful or not
  - the grade earned by the learner must be reported
  - the minimum and maximum possible grade must be reported
- The learning activities must be uniquely and consistently identified across all statements
- The time at which a learner completed a learning activity must be recorded
  - The timestamp should contain an appropriate level of specificity.
  - ie. Year, Month, Day, Hour, Minute, Second, Timezone

1.2 Input Data Retrieval

How to query an LRS via a GET request to the Statements Resource via curl. The following section contains the appropriate parameters with example values as well as the curl command necessary for making the request.[1][2][3]

[1] S is the set of all statements parsed from the statements array within the HTTP response to the Curl request(s). 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

[2] Querying an LRS will not be defined within the following Z specifications but the results of the query will be utilized

[3] If you want to query across the entire history of a LRS, omit Since and Until from the endpoint(s) and remove the associated & symbols.

```
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
```

1.3 Statement Parameters to Utilize

The statement parameter locations here are written in JSONPath. This notation is also compatable with the xAPI Z notation due to the defined hierarchy of components. Within the Z specifications, a variable name will be used instead of the $

- $\$.timestamp$
- $\$.result.success$
- $\$.result.score.raw$
- $\$.result.score.min$
- $\$.result.score.max$
- $\$.verb.id$

1.4 2018 Pilot TLA Statement Problems

The initial pilot test data supports this algorithm. This section may require updates pending future data review following iterations of the TLA testing.

1.5 Summary

1. Query an LRS via a GET request to the statements endpoint using the parameters agent, since and until

2. Filter the results to the set of statements where:
   - $\$.verb.id$ is one of:
     - http://adlnet.gov/expapi/verbs/passed
     - https://w3id.org/xapi/dod-isd/verbs/answered
     - http://adlnet.gov/expapi/verbs/completed
   - $\$.result.success$ is true
3. process the filtered data
   - extract $\$.timestamp$
   - extract the score values from $\$.result.score.raw$, $\$.result.score.min$ and $\$.result.score.max$ and convert them to the scale 0..100
   - create a pair of [$\$.timestamp$, #]

1.6 Formal Specification

1.6.1 Basic Types

$COMPLETION ::=$
$\{http://adlnet.gov/expapi/verbs/passed\} \mid$
$\{https://w3id.org/xapi/dod-isd/verbs/answered\} \mid$
$\{http://adlnet.gov/expapi/verbs/completed\}$

$SUCCESS ::= \{true\}$

1.6.2 System State

$$\begin{array}{|l}
\textit{TimelineLearnerSuccess} \\
\hline
Statements \\
S_{all} : \mathbb{F}_1 \\
S_{completion}, S_{success}, S_{processed} : \mathbb{F} \\
\hline
S_{all} = statements \\
S_{completion} \subseteq S_{all} \\
S_{success} \subseteq S_{completion} \\
S_{processed} = \{pair : (statement.timestamp, \mathbb{N})\} \\
\hline
\end{array}$$

- The set S_{all} is a non-empty, finite set and is the component *statements*
- The sets $S_{completion}$ and $S_{success}$ are both finite sets
- the set $S_{completion}$ is a subset of S_{all} which may contain every value within S_{all}
- the set $S_{success}$ is a subset of $S_{completion}$ which may contain every value within $S_{completion}$
- the set $S_{processed}$ is a finite set of pairs where each contains a *statement.timestamp* and a natural number

1.6.3 Initial System State

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{InitTimelineLearnerSuccess}} \\
\hline
\textit{TimelineLearnerSuccess} \\
\hline
S_{all} \neq \emptyset \\
S_{completion} = \emptyset \\
S_{success} = \emptyset \\
S_{processed} = \emptyset \\
\hline
\end{array}$$

- The set S_{all} is a non-empty set
- The sets $S_{completion}, S_{success}$ and $S_{processed}$ are all initially empty

1.6.4 Filter for Completion

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Completion}} \\
\hline
\textit{Statement} \\
\textit{completion} : \textit{STATEMENT} \nrightarrow \mathbb{F} \\
s? : \textit{STATEMENT} \\
s! : \mathbb{F} \\
\hline
s? = \textit{statement} \\
s! = \textit{completion}(s?) \\
\textit{completion}(s?) = \textbf{if } s?.\textit{verb.id} : \textit{COMPLETION} \\
\qquad\qquad\qquad\qquad \textbf{then } s! = s? \\
\qquad\qquad\qquad\qquad \textbf{else } s! = \emptyset \\
\hline
\end{array}$$

- The schema *Completion* inroduces the function *completion* which takes in the variable $s?$ and returns the variable $s!$
- The variable $s?$ is the component *statement*
- $s!$ is equal to $s?$ if $\$.verb.id$ is of the type $COMPLETION$ otherwise $s!$ is an empty set

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{FilterForCompletion}} \\
\hline
\Delta \textit{TimelineLearnerSuccess} \\
\textit{Completion} \\
\textit{completions} : \mathbb{F} \\
\hline
\textit{completions} \subseteq S_{all} \\
\textit{completions}' = \{ s : \textit{STATEMENT} \mid \textit{completion}(s) \neq \emptyset \} \\
S'_{completion} = S_{completion} \cup \textit{completions}' \\
\hline
\end{array}$$

- the set *completions* is a subset of S_{all} which may contain every value within S_{all}

- The set $completions'$ is the set of all statements s where the result of $completion(s)$ is not an empty set
- the updated set $S'_{completion}$ is the union of the previous state of set $S_{completion}$ and the set $completions'$

1.6.5 Filter for Success

$$
\begin{array}{|l}
\textit{Success} \\
\hline
Statement \\
success : STATEMENT \rightarrowtail \mathbb{F} \\
s? : STATEMENT \\
s! : \mathbb{F} \\
\hline
s? = statement \\
s! = success(s?) \\
success(s?) = \textbf{if } s?.result.success : SUCCESS \\
\qquad\qquad\qquad\quad \textbf{then } s! = s? \\
\qquad\qquad\qquad\quad \textbf{else } s! = \emptyset \\
\hline
\end{array}
$$

- the schema $Success$ introduces the function $success$ which takes in the variable $s?$ and returns the variable $s!$
- the variable $s?$ is the component $statement$
- $s!$ is equal to $s?$ if $\$.result.success$ is of the type $SUCCESS$ otherwise $s!$ is an empty set

$$
\begin{array}{|l}
\textit{FilterForSuccess} \\
\hline
\Delta TimelineLearnerSuccess \\
Success \\
successes : \mathbb{F} \\
\hline
successes \subseteq S_{completion} \\
successes' = \{s : STATEMENT \mid success(s) \neq \emptyset\} \\
S'_{success} = S_{success} \cup successes' \\
\hline
\end{array}
$$

- the set $successes$ is a subset of $S_{completion}$ which may contain every value within $S_{completion}$
- The set $successes'$ contains elements s of type $STATEMENT$ where $success(s)$ is not an empty set
- The updated set $S'_{success}$ is the union of the previous state of $S_{success}$ and $successes'$

1.6.6 Processes Results

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Scale}} \\
\hline
scaled! : \mathbb{N} \\
raw?, min?, max? : \mathbb{Z} \\
scale : \mathbb{Z} \rightarrow \mathbb{N} \\
\hline
scaled! = scale(raw?, min?, max?) \\
scale(raw?, min?, max?) = \\
\qquad\qquad\qquad (raw? * ((0.0 - 100.0)\ div\ (min? - max?))) + \\
\qquad\qquad\qquad (0.0 - (min? * ((0.0 - 100.0) div (min? - max?)))) \\
\hline
\end{array}
$$

- The schema $Scale$ introduces the function $scale$ which takes 3 arguments, $raw?$, $min?$ and $max?$. The function converts $raw?$ from the range $min?..max?$ to 0.0..100.0

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{ProcessStatements}} \\
\hline
\Delta TimelineLearnerSuccess \\
Scale \\
FilterStatements \\
processed : \mathbb{F} \\
\hline
processed \subseteq S_{success} \\
processed' = \{p : (\mathbb{F}_1 \#1, \mathbb{N}) \mid \\
\qquad\qquad \mathbf{let}\ \{processed_i..processed_j\} == \{s_i..s_j\} \bullet \\
\qquad\qquad i \leq n \leq j \bullet \forall s_n : s_i..s_j \bullet \exists p_n : p_i..p_j \bullet \\
\qquad\qquad first\, p_n = s_n.timestamp \wedge \\
\qquad\qquad second\, p_n = scale(s_n.result.score.raw, \\
\qquad\qquad\qquad\qquad s_n.result.score.min, \\
\qquad\qquad\qquad\qquad s_n.result.score.max)\} \\
S'_{processed} = S_{processed} \cup processed' \\
\hline
\end{array}
$$

- The operation $ProcessStatements$ introduces the variable $processed$ which is a subset of $S_{success}$ which may contain every value within $S_{success}$
- $S_{success}$ is the result of the operation $FilterStatements$
- The operation defines the variable $processsed'$ which is a set of objects p which are ordered pairs of (1) a finite set containing one value and (2) a single positive number.
- The first component of every object p, is the timestamp from the associated *statement* within *processed* ie. $s.timestamp$
- The second component of every object p is the result of the function $scale$. The score values contained within the associated *statement* s are the arguments passed to $scale$. ie $scale(s.result.score.raw, s.result.score.min, s.result.score.max)$
- The result of the operation $ProcessStatements$ is to updated the set $S_{processed}$ with the values contained within $processed'$

1.6.7 Sequence of Operations

$FilterStatements \mathrel{\widehat=} FilterForCompletion \mathbin{\text{⨾}} FilterForSuccess$

- The schema $FilterStatements$ is the sequential composition of operation schemas $FilterForCompletion$ and $FilterForSuccess$
- $FilterForCompletion$ happens before $FilterForSuccess$

$ProcessedStatements \mathrel{\widehat=} FilterStatements \mathbin{\text{⨾}} ProcessStatements$

- The schema $ProcessedStatements$ is the sequential composition of operation schemas $FilterStatements$ and $ProcessStatements$
- $FilterStatements$ happens before $ProcessStatements$

1.6.8 Return

$$
\begin{array}{|l}
\hline
\textit{Return} \\
\Xi TimelineLearnerSuccess \\
ProcessedStatements \\
S_{processed}! : \mathbb{F} \\
\hline
S_{processed}! = S_{processed} \\
\hline
\end{array}
$$

- The returned variable $S_{processed}!$ is equal to the current state of variable $S_{processed}$ after the operations $FilterForCompletion$, $FilterForSuccess$ and $ProcessStatements$

1.7 Pseudocode

Algorithm 1: Timeline of Learner Success

```
Input: S_all
Result: coll'
coll = [];
while S_all ≠ ∅ do
    foreach s ∈ S_all do
        if s.verb.id = COMPLETION then
            do
            S'_completion ← s ∪ S_completion;
            S'_all ← S_all \ s ;
            recur S'_completion, S'_all;
        else
            do
            S'_all ← S_all \ s;
            recur S'_all;
        end
    end
end
while S'_completion ≠ ∅ do
    foreach sc ∈ S'_completion do
        if sc.result.success = SUCCESS then
            do
            S'_success ← sc ∪ S_success;
            S'_completion ← S_completion \ sc;
            recur S'_success, S'_completion;
        else
            do
            S'_completion ← S_completion \ sc;
            recur S'_completion;
        end
    end
end
foreach ss ∈ S'_success do
    raw? ← ss.result.score.raw;
    max? ← ss.result.score.max;
    min? ← ss.result.score.min;
    scaled ← scale(raw?, min?, max?);
    subVec ← [ss.timestamp, scaled];
    coll' ← coll ⌢ subVec;
    recur coll'
end
return coll'
```

- The Z schemas are used within this pseudocode
- The return value coll is an array of arrays, each containing a *statement.timestamp* and a scaled score.

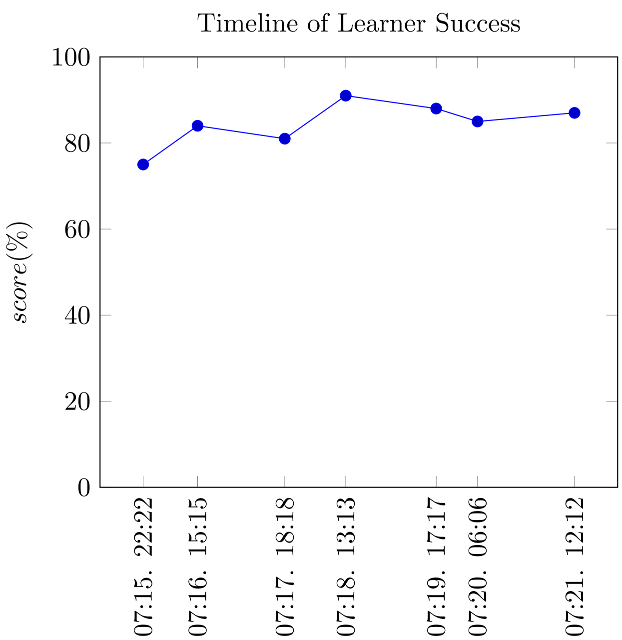
1.8 JSON Schema


```
{"type":"array",
   "items":{"type":"array",
      "items":[{"type":"string"}, {"type":"number"}]}}
```


1.9 Visualization Description

The **Timeline of Learner Success** visualization will be a line chart where the domain is time and the range is score on a scale of 0.0 to 100.0. Every subarray will be a point on the chart. The domain of the graph should be in chronological order.

1.10 Visualization prototype



1.11 Prototype Improvement Suggestions

Additional features may be implemented on top of this base specification but they would require adding aditional values to each subarray returned by the algorithm. These additional values can be retrieved via (1) performing metadata lookup within or independently of the algorithm (2) by utilizing additional xAPI statement paramters and/or (3) by performing additional computations. The following examples assume the metadata is contained within each statement available to the algorithm.

- A tooltip containing the name of an activity when hovering over a specific point on the chart
  - this would require utilizing $\$.object.definition.name$
- A tooltip containing the device on which the activity was experienced
  - this would require utilizing $\$.context.platform$
- A tooltip containing the instructor associated with a particular data point
  - this would require utilizing $\$.context.instructor$