1 Rate of Completions

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 rate of completion[1] of the various digital resources within the learning ecosystem.

1.1 Ideal Statements

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

- statements describing a learner completing an activity should[2] use the verb $http://adlnet.gov/expapi/verbs/completed$
- statements describing a learner completing an activity should report if the learner was successful or not via $\$.result.success$
- statement describing a learner completing a scored activity should report the learners score via $\$.result.score.raw$, $\$.result.score.min$ and $\$.result.score.max$
- activites 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
- statements describing a learner completing an activity should report the amount of time taken to complete the activity via $\$.result.duration$

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.[3][4][5]

[1] Completion can be defined by the presence of the verb completed or by the presence of $\$.result.completion$ set equal to true. In this algorithm, completion is defined by the presence of the verb completed regardless of $\$.result.completion$. This decision affects how statements are retrieved and filtered. In the case where completion is defined by $\$.result.completion$, the query to the LRS would not include the verb paramater and there would need to be a filtering process which looks for the presence of $\$.result.completion = \text{true}$

[2] See footnote 4

[3] See footnote 1.

[4] See footnote 2.

[5] See footnote 3.

```
Verb = "verb=http://adlnet.gov/expapi/verbs/completed"

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

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

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

endpoint = Base + Verb + "&" + 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*
- $.*object.id*

1.4 2018 Pilot TLA Statement Problems

The initial pilot test data supports the core requirements of this algorithm but completion statements only reports completion scores via $.*result.scaled* instead of $.*result.score.raw*, $.*result.score.min* and $.*result.score.max*.[6] Given that the offical 2018 pilot test is scheduled to take place on July 27th, 2018, this section may require updates pending future data review.

1.5 Summary

1. Query an LRS via a GET request to the statemetns endpoint using the paramters verb, since and until.

[6] The one potential issue with using scaled score is the calculation of scaled is not stricly defined by the xAPI specification but is instead up to the authors of the LRP. This results in the inability to reliably compare scaled scores across LRPs. if $.*result.score.raw* , $.*result.score.min* and $.*result.score.max* are reported for all questions, it becomes possible to reliably compare scores across LRPs by generating a scaled score in a consistent way.

2. group statements by their $.*object.id*

3. select time range unit for use within rate calculation. Will default to day.

4. determine the amount of time between the first and last instance of a $.*object.id* (in seconds) and divide it by the time unit. ie if the unit is minute, you would divide by 60.

5. calculate the rate by dividing the count of a group (2) by the number of time units covered by the statements (4) so that the rate is the number of completions per activity per time unit.

1.6 Formal Specification

1.6.1 Basic Types

$TIMEUNIT :== \{second\}|\{minute\}|\{hour\}|\{day\}|\{week\}|\{month\}|\{year\}$

1.6.2 System State

RateOfCompletion

$Statements$
$S_{completions} : \mathbb{F}_1$
$S_{grouped}, S_{timeunit}, S_{processed} : \mathbb{F}$

$S_{completions} = statements$
$S_{grouped} = \{byId : \text{seq}_1\ statement\}$
$S_{withRate} = \{byGroup : (\text{seq}_1\ statement, \mathbb{N})\}$
$S_{processed} = \{rate : (id, \mathbb{N}, TIMEUNIT)\}$

- The set $S_{completions}$ is a non-empty, finite set and is the component *statements* which contains the results of the query to the LRS.
- The sets $S_{grouped}$, $S_{withRate}$ and $S_{processed}$ are all finite sets
- the set $S_{grouped}$ is a finite set of objects *byId* which are non-empty, finite sequences of the component *statement*
- the set $S_{withRate}$ is a finite set of objects *byGroup* which are ordered pairs of non-empty, finite sequences of the component *statement* and a natural number
- the set $S_{processed}$ is a finite set of objects *rate* where each contains the component *id*, a natural number and the type $TIMEUNIT$

1.6.3 Initial System State

$InitRateOfCompletion$

$RateOfCompletion$
$T : TIMEUNIT$

$S_{completions} \neq \emptyset$
$S_{grouped} = \emptyset$
$S_{withRate} = \emptyset$
$S_{processed} = \emptyset$
$T = \{day\}$

- The set $S_{completions}$ is a non-empty set which contains the results of the GET request(s) to the LRS
- The sets $S_{grouped}$, $S_{withRate}$ and $S_{processed}$ are all initially empty
- the variable T has the type $TIMEUNIT$ and the value $\{day\}$

1.6.4 Calculate Rate

$IsoToUnix$

$convert : \mathbb{F}_1 \rightarrow \mathbb{N}\#1$
$c? : \mathbb{F}_1$
$c! : \mathbb{N}\#1$

$c! = convert(c?)$

- The schema $IsoToUnix$ introduces the function *convert* which takes in a finit set of one thing (a timestamp) and converts it to a single natural number.
- the purpose of this function is to convert an ISO 8601 timestamp to the Unix epoch. The concrete definition of the conversion is outside the scope of this document
  - The Unix epoch is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap seconds.

$$
\begin{array}{|l}
\hline
\textit{CalcRateByUnit} \\
\textit{Statement} \\
\textit{IsoToUnix} \\
\textit{CountPerGroup} \\
\textit{unit?} : TIMEUNIT \\
s? , s! : \mathbb{F} \\
r : \mathbb{N} \\
\textit{rate} : (\mathbb{F}, TIMEUNIT) \to \mathbb{F} \\
\hline
\textit{unit?} = \{\textit{second}\} \Rightarrow 1 \lor \{\textit{minute}\} \Rightarrow 60 \lor \{\textit{hour}\} \Rightarrow 3600 \lor \\
\qquad \{\textit{day}\} \Rightarrow 86400 \lor \{\textit{week}\} \Rightarrow 604800 \lor \\
\qquad \{\textit{month}\} \Rightarrow 2629743 \lor \{\textit{year}\} \Rightarrow 31556926 \\
s? = \{g : \textit{seq}_1 \textit{statement}\} \\
s! = \textit{rate}(s? , \textit{unit?}) \\
s! = \{s : (g, r) \mid \forall g_n : g_i..g_j \bullet i \leq n \leq j \bullet \exists s_n : (g_n, r_n) \bullet \\
\qquad r_n = \textit{count}(g_n) \div ((\textit{convert}(\textit{last}\, g_n.\textit{timestamp}) - \textit{convert}(\textit{head}\, g_n.\textit{timestamp})) \div \textit{unit?})\} \\
\hline
\end{array}
$$

- The schema $\textit{CalcRateByUnit}$ introduces the function $\textit{rate}$ where the input $s?$ is a set of objects g which are each a non-empty, finite sequence of statements and the input $\textit{unit?}$ represents a unit of time.
- for every g_n within the range $g_i..g_j$, there exists an associated object s_n which is an orderd pair of (g_n, r_n) where r_n is equal to the number of items within g_n divided by the number of $\textit{unit?}$s within the time range of $\textit{last}\, g_n.\textit{timestamp} - \textit{head}\, g_n.\textit{timestamp}$
- the output of the function $\textit{rate}$ is $s!$, the set of all s_n

1.6.5 Processes Results

$$
\begin{array}{|l}
\hline
\textit{AggergateCompletionStatements} \\
\Delta RateOfCompletion \\
GroupByActivityId \\
CalcRateByUnit \\
grouped, processed, withRate : \mathbb{F} \\
r : \mathbb{N} \\
T? : TIMEUNIT \\
\hline
T? = \{day\} \\
grouped = \emptyset \\
grouped' = group(S_{completions}) \\
S'_{grouped} = S_{grouped} \cup grouped' \\
withRate \subseteq S'_{grouped} \\
withRate' = rate(withRate, T?) \\
S'_{withRate} = withRate' \cup S_{withRate} \\
processed \subseteq S'_{withRate} \\
processed' = \{p : (id, r, T?) \mid \\
\qquad \textbf{let } \{processed_i..processed_j\} == \{b_i..b_j\} \bullet \\
\qquad \forall b_n : b_i..b_j \bullet i \leq n \leq j \bullet \exists p_n : (id_n, r_n, T?) \bullet \\
\qquad id_n = (head\,(first\,b_n)).object.id \land \\
\qquad r_n = (second\,b_n)\} \\
S'_{processed} = processed' \cup S_{processed} \\
\hline
\end{array}
$$

- The schema *AggergateCompletionStatements* outlines how to calculate the rate of completion per $.*object.id* per *second*|*minute*|*hour*|*day*|*week*|*month*|*year*
  1. $S'_{grouped}$ is the result of grouping the statements within $S_{completions}$ by their $.*object.id*
  2. The groups from (1) are passed to the function *rate* with the variable $T?$ which controls the unit of time, ie per day vs per week
  3. the result of (2) is then processed to create a triplet of $.*object.id*, rate, unit of time for all unique $.*object.id* within $S_{completions}$

1.6.6 Return

$$
\begin{array}{|l}
\hline
\textit{ReturnAggergateCompletionStatements} \\
\Xi RateOfCompletion \\
AggergateCompletionStatements \\
S_{processed}! : \mathbb{F} \\
\hline
S_{processed}! = S_{processed} \\
\hline
\end{array}
$$

- The return value $S_{processed}!$ is equal to $S_{processed}$ after the operation described by *AggergateCompletionStatements*

1.7 Pseudocode

Algorithm 1: Rate of Completions

Input: $S_{completed}$, $timeUnit$
Result: $ratePerObjTu'$
$context = \{\}$;
$ratePerObjTu = []$;
while $S_{completion} \neq \emptyset$ **do**
  foreach $s \in S_{completion}$ **do**
    $id \leftarrow s.object.id$;
    $ts \leftarrow convert(s.timestamp)$;
    if $id \notin context$ **then**
      do
      $times = [ts]$;
      $context' \leftarrow \{id : times\}$;
      $S'_{completion} \leftarrow S_{completion} \setminus s$;
      recur $context', S'_{completion}$;
    else
      do
      $times' \leftarrow context.id \frown ts$;
      $context' \leftarrow \{id : times'\}$;
      $S'_{completion} \leftarrow S_{completion} \setminus s$;
      recur $context', S'_{completion}$;
    end
  end
end
foreach $k \in context'$ **do**
  $allTs \leftarrow context'.k$;
  $totalDuration \leftarrow max(allTs) - min(allTs)$;
  $totalCount \leftarrow count(allTs)$;
  $rate \leftarrow totalCount \div (totalDuration \div timeUnit)$;
  $subVec = [k, rate, timeUnit]$;
  $ratePerObjTu' \leftarrow ratePerObjTu \frown subVec$;
  recur $ratePerObjTu'$;
end
return $ratePerObjTu'$

- Values from Z schemas are used within this pseudocode
- the result of the algorithm is an array of arrays where each subarray contains a *statement.object.id*, the *rate* and the *timeUnit* used to calculate *rate*.

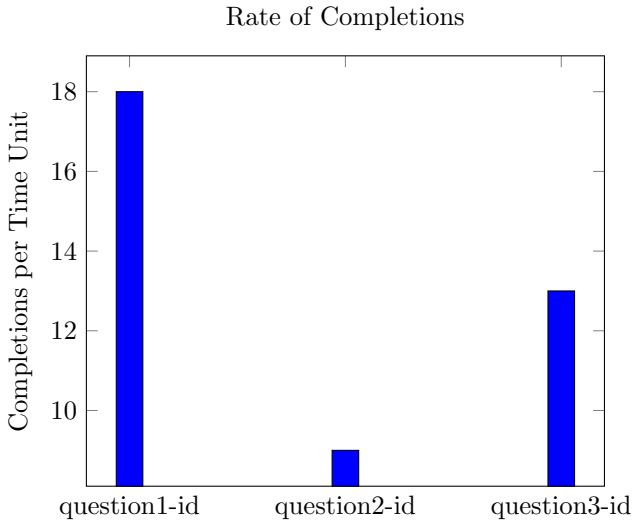
1.8 JSON Schema

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

1.9 Visualization Description

The **Rate of Completions** visualization will be a bar chart where the domain consists of *statement.object.id* and the range is a number greater than 0 (the rate of completions for that *statement.object.id*). Every subarray within the array *ratePerObjTu* will be a grouping within the bar chart. The pseudocode specifies an input paramter *timeUnit* which controls the calculation of the rate (range of the visualization). *timeUnit* could be per minute, per day, per week, etc.

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.

- use *statement.object.definition.name* instead of *statement.object.id* for x axis label
- populate a tooltip with the people who have completed the activity. This could also include the number of times they have completed it.
- populate a tooltip with the breakdown of which devices or platforms the activity was completed on. This would require the device type or platform to be reported within *statement.context.platform*
- populate a tooltip with the breakdown of percentage successful for all completions of the activty. This would require *statement.result.success*
- populate a tooltip with the breakdown of scores earned (if appliciable) for the completions. This would require *statement.result.score.raw*, *statement.result.score.min* and *statement.result.score.max*
- populate a tooltip with the competency assocaited with the completed activities. The competency should be reported via *statement.context.contextActivities*
- populate a tooltip with the average duration spent to reach completions. This would require *statement.result.duration* to be reported.