Introduction

some stand in intro text

1 xAPI Formal Specification

The current formal specification only defines xAPI statements abstractly within the context of Z. A concrete definition for xAPI statements is outside the scope of this document.

1.1 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

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

- 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 type for an xAPI data point

$[AGENT, GROUP]$

- Basic types for Agents and collections of Agents

1.2 Id Schema

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

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

1.3 Schemas for Agents, Groups and Actors

Agent

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

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

Member

$$
\begin{array}{l}
Agent \\
member : \mathbb{F}_1 \\
\hline
member = \{a : AGENT \mid \forall a_n : a_i..a_j \bullet i \leq n \leq j \bullet a = agent\}
\end{array}
$$

- 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*

Group

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

- 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*

Actor

$$
\begin{array}{l}
Agent \\
Group \\
actor : AGENT \lor GROUP \\
\hline
actor = agent \lor group
\end{array}
$$

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

1.4 Verb Schema

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

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

1.5 Object Schema

$$
\begin{array}{|l}
\hline
Extensions \\
\hline
extensions, extensionVal : \mathbb{F}_1 \\
extensionId : \mathbb{F}_1 \, \#1 \\
\hline
extensions = \{e : (extensionId, extensionVal) \mid \forall e_n : e_i..e_j \bullet i \leq n \leq j \bullet \\
\qquad (extensionId_i, extensionVal_i) \lor (extensionId_i, extensionVal_j) \land \\
\qquad (extensionId_j, extensionVal_i) \lor (extensionId_j, extensionVal_j) \land \\
\qquad extensionId_i \neq extensionId_j\} \\
\hline
\end{array}
$$

- 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

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

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

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

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

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

- The schema $Object$ introduces the component $object$ which is a non-empty, finite set of either id, id and $objectType$, id and $objectTypeA$, 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

1.6 Result Schema

Score

$$score : \mathbb{F}_1$$
$$scaled, min, max, raw : \mathbb{Z}$$

$$scaled = \{n : \mathbb{Z} \mid -1.0 \leq n \leq 1.0\}$$
$$min = n < max$$
$$max = n > min$$
$$raw = \{n : \mathbb{Z} \mid min \leq n \leq max\}$$
$$score = \mathbb{P}_1\{scaled, raw, min, max\}$$

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

Result

$$Score$$
$$Extensions$$
$$success, completion, response, duration : \mathbb{F}_1 \#1$$
$$result : \mathbb{F}_1$$

$$success = \{true\} \lor \{false\}$$
$$completion = \{true\} \lor \{false\}$$
$$result = \mathbb{P}_1\{score, success, completion, response, duration, extensions\}$$

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

1.7 Context Schema

Instructor

$$Agent$$
$$Group$$
$$instructor : AGENT \lor GROUP$$

$$instructor = agent \lor group$$

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

Team

$$Group$$
$$team : GROUP$$

$$team = group$$

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

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

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

- The notation $object \setminus agent$ represents the component $object$ except for its subcomponent $agent$

1.8 Timestamp and Stored Schema

$$
\begin{array}{|l}
\hline
Timestamp \\
\hline
timestamp : \mathbb{F}_1 \#1 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
Stored \\
\hline
stored : \mathbb{F}_1 \#1 \\
\hline
\end{array}
$$

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

1.9 Attachements Schema

Attachments

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

- The schema *Attachements* introduces the componenet *attachements* which is a non-empty, finite set of the component *attachement*
- The component *attachment* is a non-empty, finite set of the componenets *usageType*, *display*, *contentType*, *length*, *sha*2 with optionally *description* and/or $fileUrl$

1.10 Statement and Statements Schema

Statement

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

- 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* allows for subcomponent of *statement* to refrenced via the . (selection) operator

Statements

$$
\begin{array}{l}
Statement \\
IsoToUnix \\
statements : \mathbb{F}_1 \\
\hline
statements = \{s : statement \mid \forall s_n : s_i..s_j \bullet i \leq n \leq j \\
\qquad \bullet convert(s_i.timestamp) \leq convert(s_j.timestamp)\}
\end{array}
$$

- The schema *Statements* introduces the component *statements* which is a non-empty, finite set of the component *statement* which are in chronological order.

2 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.

2.1 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

2.2 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
```

2.3 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*

2.4 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.

2.5 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*, #]

2.6 Formal Specification

2.6.1 Basic Types

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

$SUCCESS :== \{true\}$

2.6.2 System State

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

- 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

2.6.3 Initial System State

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

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

2.6.4 Filter for Completion

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

- 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

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

- 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 $completions'$

2.6.5 Filter for Success

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

- 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

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

- 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 $successes'$

2.6.6 Processes Results

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

- 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

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

- 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 arugments 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*$'$

2.6.7 Sequence of Operations

$FilterStatements \mathrel{\widehat=} FilterForCompletion \fatsemi FilterForSuccess$

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

$ProcessedStatements \mathrel{\widehat=} FilterStatements \fatsemi ProcessStatements$

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

2.6.8 Return

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

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

2.7 Pseudocode

Algorithm 1: Timeline of Learner Success

Input: S_{all}
Result: $coll'$
$coll = []$;
while $S_{all} \neq \emptyset$ **do**
  foreach $s \in S_{all}$ **do**
    if $s.verb.id = COMPLETION$ **then**
      do
      $S'_{completion} \leftarrow s \cup S_{completion}$;
      $S'_{all} \leftarrow S_{all} \setminus s$;
      recur $S'_{completion}, S'_{all}$;
    else
      do
      $S'_{all} \leftarrow S_{all} \setminus s$;
      recur S'_{all};
    end
  end
end
while $S'_{completion} \neq \emptyset$ **do**
  foreach $sc \in S'_{completion}$ **do**
    if $sc.result.success = SUCCESS$ **then**
      do
      $S'_{success} \leftarrow sc \cup S_{success}$;
      $S'_{completion} \leftarrow S_{completion} \setminus sc$;
      recur $S'_{success}, S'_{completion}$;
    else
      do
      $S'_{completion} \leftarrow S_{completion} \setminus sc$;
      recur $S'_{completion}$;
    end
  end
end
foreach $ss \in S'_{success}$ **do**
  $raw? \leftarrow ss.result.score.raw$;
  $max? \leftarrow ss.result.score.max$;
  $min? \leftarrow ss.result.score.min$;
  $scaled \leftarrow scale(raw?, min?, max?)$;
  $subVec \leftarrow [ss.timestamp, scaled]$;
  $coll' \leftarrow coll \frown subVec$;
  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.

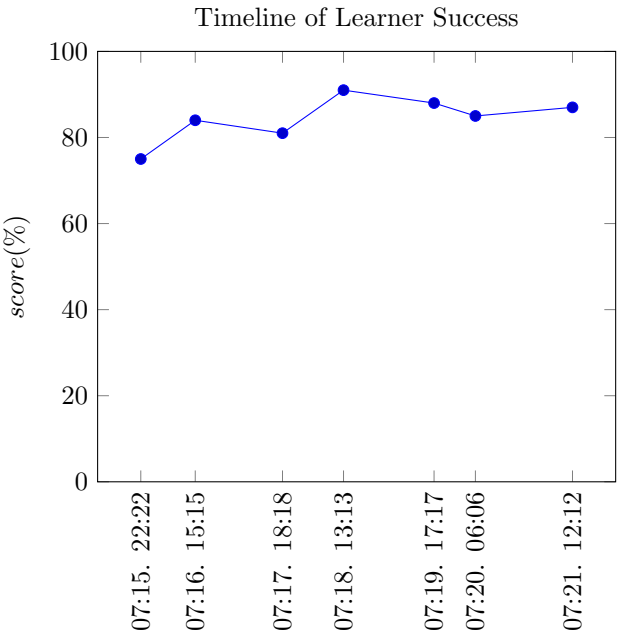
2.8 JSON Schema

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

2.9 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.

2.10 Visualization prototype



2.11 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 $\$.context.instructor$

Appendex A: Visualization Exemplars

Appendex A includes a typology of data visualizations which may be supported within DAVE workbooks. These visualizations can either be one to one or one to many in regards to the algorithms defined within this document. Future iterations of this document will increasingly include these typologies within the domain-question template exemplars.

Line Charts

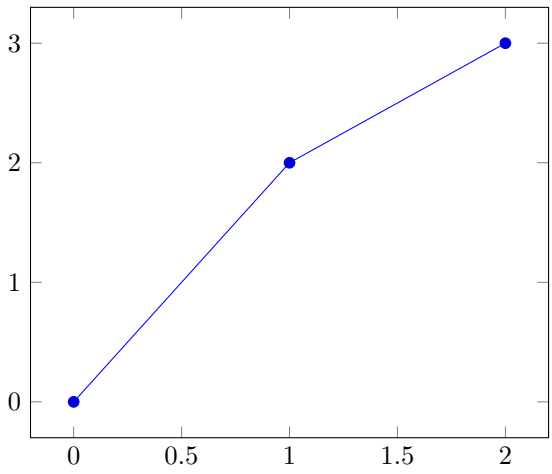


Figure 1: Line Chart

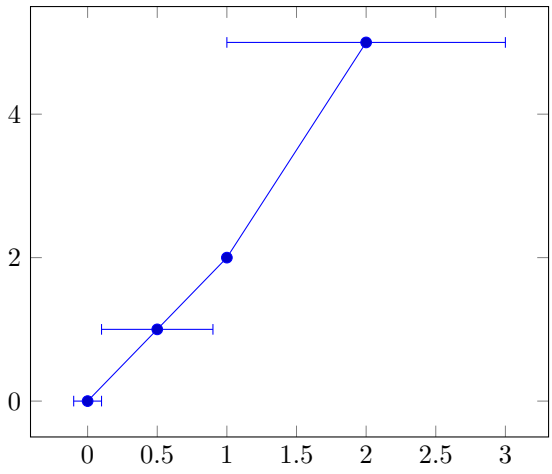


Figure 2: Line Chart with Error

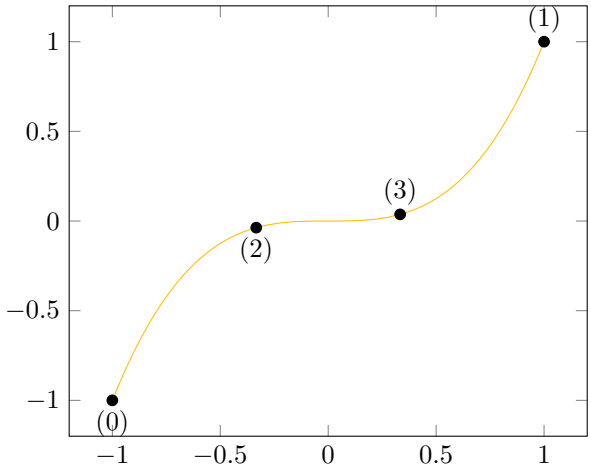


Figure 3: Spline Chart

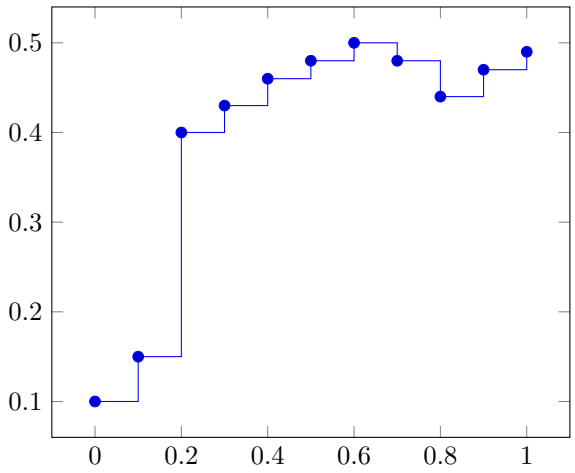


Figure 4: Quiver Chart

Grouping Charts

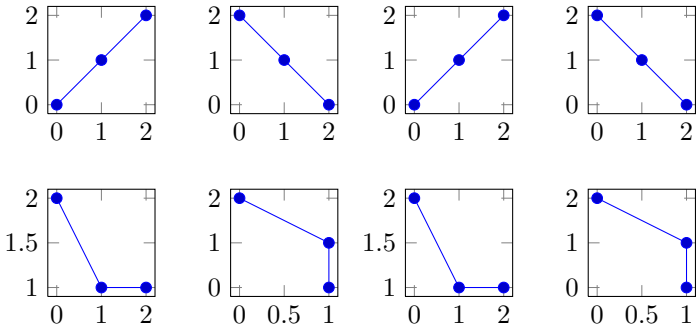


Figure 5: Grouped Line Charts

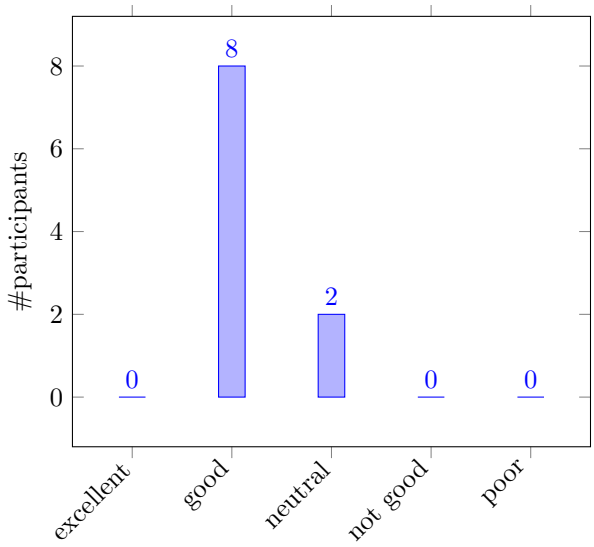


Figure 6: Histogram

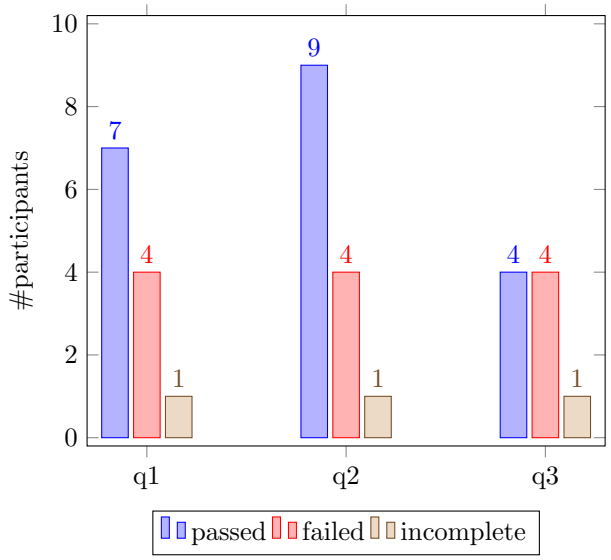


Figure 7: Bar Chart

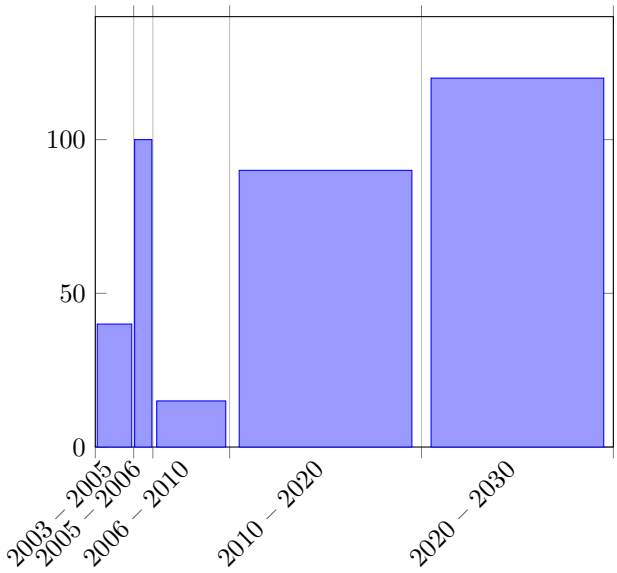


Figure 8: Bar Chart Grouped by Time Range

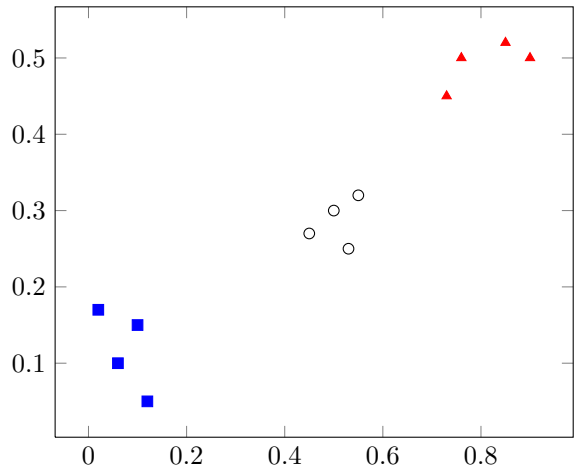


Figure 9: Scatter Plot

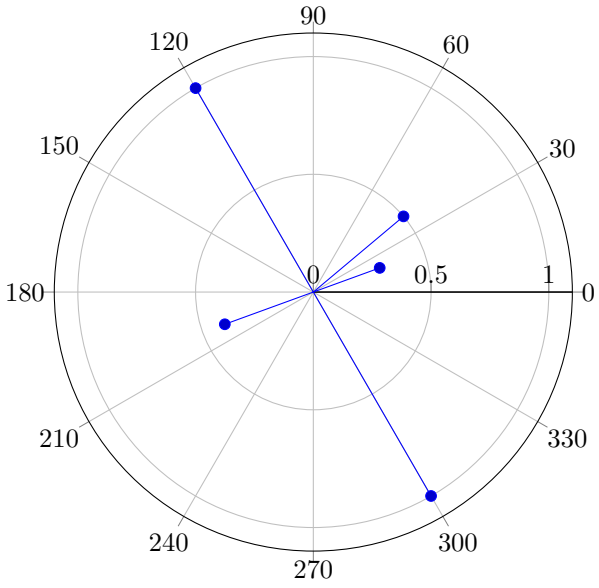


Figure 10: Polar Chart

Specialized Charts

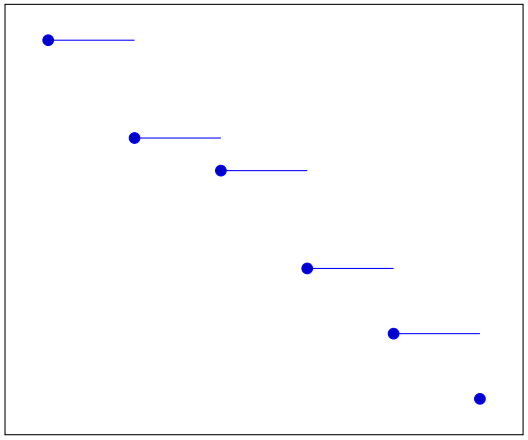


Figure 11: Gantt Chart

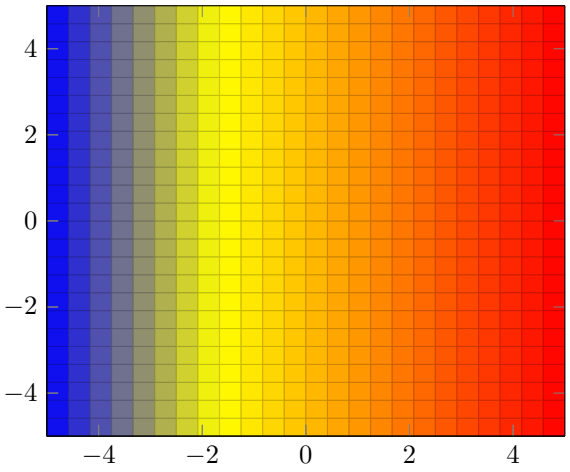


Figure 12: Heat Map

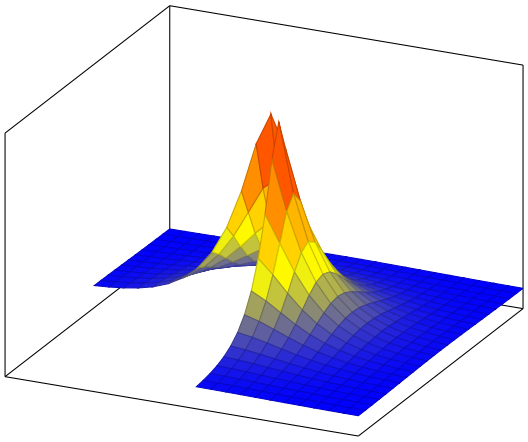


Figure 13: 3D Plot

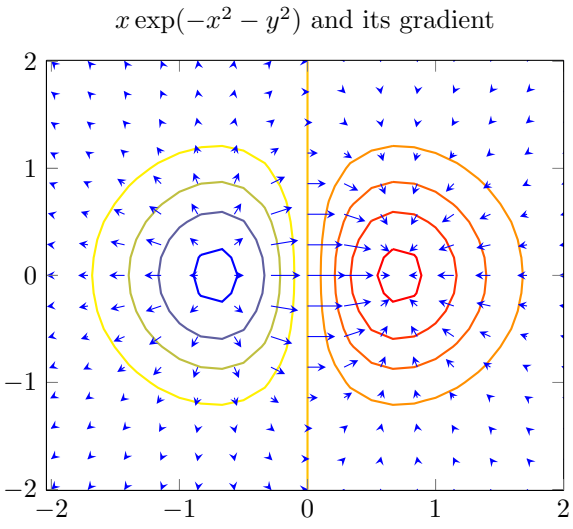


Figure 14: Gradient Plot