PART 2

xAPI Implementation

Once you have a solid understanding of xAPI, use this technical guide to develop your own xAPI implementation.

Watershed

PART 2: Let's Get Technical

How do I implement xAPI?

When you're developing your own xAPI implementation, there are two streams of work—xAPI statement design and API communication—which can be completed alongside one another. From there, simply check your data to ensure a successful implementation. This technical guide is separated into three parts:

1) xAPI Statement Design
2) API Communication
3) Check the Data

What's xAPI? *Learn more about xAPI before developing your own implementation. Read Part 1 of this eBook series now.*

1) xAPI Statement Design

After you've identified the events and data to be captured, match them to the properties of an xAPI statement. This task should be undertaken by somebody with an understanding of both the data requirements of the implementation and of xAPI statement structure.

IMPORTANT: *There are detailed guides explaining the structure of xAPI statements, and you should work through these resources carefully before developing your own xAPI implementation.*

A good way to document statement design is by creating an example statement for each event, plus supporting notes explaining the properties used. Creating examples (rather than listing properties in a table or some other method) helps you think through all the properties required and means you can use your example statements as test data before you complete your implementation. And you need supporting notes because a single example can't illustrate the range of possible values that might be used.

References

- Anatomy of a xAPI Statement
- xAPI Specification Part 2

PART 2: Let's Get Technical

Learner Identity

The learner identifier is important because many of the benefits of learning analytics rely on being able to follow an individual across multiple systems.

This applies even if you're using anonymous data and aren't interested in the actions of individuals; to answer questions such as, "How do successful people learn?", for example, you must be able to match learning data to success data.

xAPI Best Practices

As part of designing xAPI statements, you must consider two questions at an organizational level:

1) How will you identify learners?
2) How will you structure activity IDs?

This means you either need to use the same identifier across all tracked systems, or you need to use a collection of identifiers that can be associated together in your Learning Record Store (LRS).

While xAPI defines four types of identifier—mbox, account, mbox_sha1sum, and openid—most implementations to date use either account identifiers or email addresses. There's no good reason why mbox_sha1sum (a hash generated from the learner's email address) shouldn't be used, but it normally isn't in practice.

Learners often don't have relevant openids to use. So, account identifiers, such as employee IDs, that are used consistently across all systems are the preferred route. When no such employee IDs exist, email addresses are a good secondary option because most people have them.

Employee IDs are often preferable because:

- People in non-computer based jobs may not have email addresses, or they share an email address between a group of people.
- If an employee leaves a company and then another person with the same name is hired, the email address of the original employee may reused. In this case, the former employee and current employee effectively share the same email address (albeit, separated in time).
- People can revise their email addresses after a name change (e.g., after getting married).

PART 2: Let's Get Technical

Learner Identity *(continued)*

Changes to email addresses can be handled relatively easily by associating both the old and new email address to the learner. Shared email addresses are more problematic because it's not possible to distinguish people who share the same "unique" identifier. You should evaluate the significance of these issues when choosing which identifier to use.

When using an account identifier, the actor will look something like this:

```
"actor": {
    "name": "John Doe",
     "account": {
     "name": "12345",
      "homePage": "https://hr.your-org.com"
      }
    }
```

The account name is simple: It's John Doe's employee ID that uniquely identifies him within your organization. The xAPI actor is supposed to be universally unique, however. This protects you in case you need to merge learning data (e.g., your organization merges with another one).

In this scenario, there could be another person with the same employee ID number as John, so you must include something to identify this is John's employee ID number as defined by your organization rather than somebody else's ID number.

The account homePage provides that uniqueness. It's a URL owned by your organization, so it won't be used by anybody else. This doesn't have to be a live URL, it just has to be a domain that you own and control.

In some cases, you may find that you need to use different identifiers for different systems and then link those identifiers together in your LRS. This is relatively common and supported by most good LRSs.

PART 2: Let's Get Technical

Activity IDs

An activity ID is a unique identifier for a particular activity. In xAPI, an activity could be anything you want to measure (i.e., taking a quiz, filling out a survey, watching a video, etc.).

A common—and often problematic—mistake with xAPI implementations is getting the activity ID wrong. There are three ways people make this mistake:

1) The ID format is incorrect.
2) Multiple IDs are used when a single ID should be used.
3) A single ID is used when multiple IDs should be used.

Recommended Reading: Check out our detailed guide, "Get the Activity ID Right" for more about these issues, including helpful examples.

Organizations should create an internal system to ensure that activity IDs are unique and consistent. For example, this could include:

- An internal registry of activity IDs, perhaps in the form of a shared document or intranet page
- Quality control processes to ensure good activity IDs are used
- A naming convention for structuring activity IDs Different base URLs used by different teams, so each team can be responsible for the uniqueness of their own IDs

Activity ID Example

An organization (example.com) might create activity IDs using a structure such as:

https://example.com/team-code/name-of-authoring-tool/name-of-course/unqiue-course-id

Each team would be responsible for ensuring the uniqueness of activity IDs starting with their team codes.

PART 2: Let's Get Technical

2) API Communication

In addition to designing the xAPI statements, you also need to make the connection between your application and your LRS to send the statements.

This more technical side of xAPI is in some ways more straightforward than statement design because there are existing code libraries for all the major programming languages. Simply download the code library and configure your application with xAPI endpoint and authentication details.

In the rare event that you're using a programming language that isn't covered, or you need to contribute a change to a code library, the xAPI Specification Part 3: Communication explains the requirements of communication between the LRS and client.

Server-side tracking, best tracking

If you have a choice between sending the data from the learner's browser using client-side JavaScript versus sending the data server side from a web server, server-side code is always the best option. Server-side tracking is less vulnerable to both security and connectivity issues.

Batch statements for maximum performance

It's good practice to collect and send xAPI statements in batches for maximum performance.

If sending a high volume of statements, no more than 500 statements per batch is recommended.

With smaller volumes of statements, you may wish to send smaller batches to avoid too much of a delay between when the data is generated and when it appears in your LRS.

PART 2: Let's Get Technical

Handle error codes appropriately

For a robust xAPI implementation, build your implementation to handle failed requests (e.g., resending the request as individual statements to identify the problem statement, logging the error, or alerting a human). Error codes in the 400 range normally require human intervention, whereas errors in the 500 range may be resolved by trying again later.

Have code in place to keep statements and resend them later to avoid losing data in case of a connection error or LRS downtime. In particular, your application should be built to handle the following error codes when sending statements.

Error Codes

200 OK or 204 No Content. This indicates a successful request.

400 Bad Request. There's something wrong with the structure of one or more of the statements or with the request package itself. Normally, the response will state the issue.

401 Unauthorized. The authorization header doesn't match an authorized set of credentials.

403 Forbidden. The credentials match, but those credentials are not authorized to make that request.

404 Not Found. Most likely, the endpoint is configured incorrectly. Or, there's a poor internet connection or other network issue.

413 Request Entity Too Large. The request was larger than allowed by the LRS. Try breaking the statements into smaller batches.

429 Too Many Requests. Your application has sent too many requests in a short span of time. Try sending statements in batches.

500 Internal Server Error. These responses may happen in response to bad requests that are wrong in ways that your LRS's validation has missed.

503 Service Unavailable. Indicates a period of downtime or overloaded servers.

504 Gateway Timeout. Indicates a period of downtime or overloaded servers.

Handle error codes appropriately. *(continued)*

As a rule of thumb, an error code starting in 4 suggests an issue on the client side. An error code starting in 5 suggests an issue on the LRS side. These codes should be accompanied by helpful error messages. Different LRSs provide different levels of detail in error messages and testing with multiple LRSs can be helpful when debugging an issue. Some LRSs also include functionality to log errors.

References

- Code Libraries
- xAPI Specification Part 3

Implementing xAPI in your authoring tool

The following guidelines apply across many authoring tools when implementing xAPI. Guides by authoring tool vendors, LRS vendors, and e-learning consultants also exist for various types of tools.

Use the latest version.

Many authoring tools regularly update and enhance their xAPI-tracking capabilities. Make sure you're patched to the latest version of the tool to ensure that your courses benefit from these fixes.

Using the latest versions also means you have fixes and enhancements for other features of the tool unrelated to xAPI tracking.

If you're using a cloud-based tool, you should receive automatic upgrades to the latest versions—whereas you'll need to manually upgrade desktop software.

Give slides and objects good names.

When viewing tracking data, it's more insightful to know the learner interacted with the slide titled "Implementing xAPI in your authoring tool," rather than "Slide 14."

Similarly, it's more insightful to know the learner dragged the photo of the cat to the Furry Animals Dropzone than knowing they dragged Picture 8 to Dropzone 4.

Therefore, use descriptive names for course slides and objects so you can easily see what's going on later. Detailed names can also help benefit visually impaired learners who may rely on them.

PART 2: Let's Get Technical

Implementing xAPI in your authoring tool *(continued)*

Publish for "xAPI" or "Tin Can."

When you publish the course, it's important to select the xAPI option from the authoring tool's publish settings, rather than SCORM or AICC. In some cases, the xAPI option may be called "Tin Can" instead.

Some tools may have additional settings that need to be configured as well; see your authoring tool's documentation for details.

Complete the course information.

The tool usually asks you to complete the course name and description, as well as enter an activity ID—and you should complete all of these fields. The activity ID is most important field (Read the section on activity IDs under xAPI Best Practices).

Some tools automatically add "http://" to your entry in the activity ID field. If your tool does this, remove it when you enter your activity ID to avoid having "http://" appear twice in statements.

If the authoring tool always adds "http://" at the start, it's not possible to have your activity IDs start with "https://" when using these tools.

Edit carefully after release.

If you need to revise a course after learners have started using it, you should:

- back up your project file before making any changes,
- make changes carefully, and
- test the new version with your reports before releasing it to learners.

Deleting and recreating slides and objects could result in their respective identifiers accidentally changing, which could adversely affect your data and reports.

Test your courses.

You should always test the tracking data from your courses before releasing them to learners.

You will need to either launch the course from a Learning Management System (LMS) that can support xAPI and send the data to your LRS, or use a launch wrapper.

PART 2: Let's Get Technical

Implementing xAPI in your authoring tool *(continued)*

Implement custom tracking.

In some cases, you might want to extend the tracking included in your authoring tool with additional tracking. You can do this in two simple steps:

1) Incorporate TinCanJS in your package.

Some authoring tools may already include TinCanJS as standard. If not, you'll need to include TinCanJS in your published packages. You can download the latest version of TinCanJS from GitHub and drop tincan- min.js from the build folder into the root directory of the published package.

You'll also need to link to TinCanJS from the published HTML file (this might be called something such as index.html or story.html) by adding the following code:

```
<script src="tincan-min.js"></script>
```

2) Embed statement sending code in your course.

Many authoring tools include functionality to execute JavaScript when certain events trigger in the course. Drop the following JavaScript into that feature:

```
var tincan = new TinCan ({url: window.location.href});
tincan.sendStatement(
 {
  verb: {
   id: "http://example.com/verbs/some-verb"
  },
  object: {
   id: "http://your-organizations.website/tincanapi/activities/unique-id-for-action"
  }
 }
);
```

Implementing xAPI in your authoring tool *(continued)*

Replace the statement in this code with the statement you design. (See xAPI Statement Design.) Note the statement doesn't include the actor property—it'll be added by TinCanJS based on the launch parameters used to launch the course. When using custom tracking, review the JSON statements as described in Reviewing xAPI Statements, even if the authoring tool used is a certified data source.

References

- Get started with xAPI in my content

Launching xAPI content packages

A history of launch in xAPI

xAPI defines the communication of learning records to and from an LRS. Unlike previous learning standards such as SCORM and AICC, xAPI doesn't define a mechanism for packaging content. That's because many xAPI applications don't have anything to do with packaged e-learning content. Rather, many use server-side tracking to record experiences of learners interacting with web applications.

Rustici Software, the authors of the original "Tin Can API" draft specification, however, created a companion document titled *Incorporating a TinCan LRS into an LMS*. It describes a process for packaging "Tin Can" content and launching it from a system, such as an LMS, with a similar user experience to how SCORM courses were previously loaded into LMSs.

This method of packaging and launching was intended to be temporary until a formal specification was created. It doesn't have a name, so we'll call it "Tin Can Launch." The expected formal specification, which became known as cmi5, took time to arrive. In the meantime, the "temporary" packaging and launch method became widely adopted amongst both authoring tools and LMSs that adopted xAPI.

cmi5 was released in July 2016, more than three years after Tin Can Launch was published. It's built on top of the Tin Can Launch process with changes and additional features, and is considered more robust than Tin Can Launch. It's implementation also is more involved because of these extra features. Because adoption of cmi5 is low compared to Tin Can Launch, it's usually more useful to implement Tin Can Launch first and then extend and adapt that implementation to support cmi5.

The rest of this section relates to the Tin Can Launch method rather than cmi5. See the cmi5 specification for implementation requirements.

References

- Incorporating a TinCan LRS into an LMS
- cmi5 specification

PART 2: Let's Get Technical

Launching xAPI content packages *(continued)*

Packaging

When you select the Tin Can or xAPI publish options in most e-learning course authoring tools, they will produce a ZIP package following the process outlined in the Tin Can Launch document. This means that they contain a tincan.xml file in the root directory of the package. This file contains metadata about the course and a URL to launch the course. If you're using an authoring tool, you shouldn't need to edit the tincan.xml file.

If you do need to edit the file or create your own, a full schema for the xml file can be found here and examples can be found in the prototypes hosted on experienceapi.com.

Here's the file from the Golf Course prototype, which you can use as the basis of your own packages.

```
<?xml version="1.0" encoding="utf-8" ?>
<tincan xmlns="http://projecttincan.com/tincan.xsd">
 <activities>
  <activity id="http://id.tincanapi.com/activity/tincan-prototypes/golf-example"
type="http://adlnet.gov/expapi/activities/course">
   <name>Tin Can Golf Example</name>
   <description lang="en-US">An overview of the sport of golf.</description>
   <launch lang="en-us">index.html</launch>
  </activity>
 </activities>
</tincan>
```

You should edit the activity ID, activity type, name, description, and launch fields with your own values. See the xAPI Statement Design section for more information about choosing the right values for these fields, especially the activity ID. The information included in this file should match the activity ID and definition sent in statements by your course.

The launch URL can be a relative URL pointing to a file inside the package, or an absolute URL pointing to a file hosted elsewhere; it's perfectly valid for the package to contain only the tincan.xml file with all other files served from another site.

Launching xAPI content packages *(continued)*

Launch

xAPI packages also should collect data about the learner and LRS from the system launching the package. Systems launching xAPI packages, such as LMSs, need to provide this data. The learner and LRS information is passed via a querystring. This is detailed in the Launch section of *Incorporating a TinCan LRS into an LMS*. The resulting URL looks like this (expect without line breaks and the values should be URL encoded):

```
http://example.com/
?endpoint=https://example.lrs.com/xapi/
&auth=Basic QWxhZGRpbjpPcGVuU2VzYW1l
&actor={ "name" : "Example Learner", "mbox" : "mailto:email@example.com" }
&registration=dbccef44-e4bf-47b4-a3f6-179889ea35d0
&activity_id=http://example.com/activity
```

- The endpoint is the endpoint of the LRS to send the data to.
- The auth is a Base 64 encoded version of the string key:secret where the key and secret are the credentials provided by your LRS.
- The actor is a JSON-encoded, xAPI-conformant actor object representing the learner. Note that the example in *Incorporating a TinCan LRS into an LMS* conforms to an old pre-release version of xAPI and should not be copied.
- The registration parameter is optional. It contains a UUID representing a registration or session for the course. Some courses will save bookmarking data against a particular registration, so using the same registration UUID the next time the content is launched will cause the learner to return where they left off; using a new registration UUID will give them a clean start. If not included, some courses will still store bookmarking data against a blank registration UUID, and the learner will always be returned to where they left off.
- The activity_id parameter also is technically optional but is required by some popular authoring tools. It defines the activity ID of the course and can be read directly from the tincan.xml file (where one exists).

Launching xAPI content packages *(continued)*

To help with launching content, you can use this xAPI launch tool that plays the role of an LMS but without any authentication or security. Rather than logging in, learners simply enter their names and email addresses, and the tool trusts they are who they claim. Unless you trust everybody with access to the course, this tool should be used only for testing or as a reference example when developing a more secure launch implementation in an LMS or similar.

In cases such as testing or where access to the course is secure, these files can be used to create a standalone self-launching course. In this instance, you'll need to drop the files into the folder with your package and either use an existing tincan.xml file contained in the package or update the example.

LRS details should be configured in the top of the launch.js file. You can then link the users to the launch.html file to enter their names and email addresses. You can customize the launch.html with your brand styling, imagery, and instructions as required.

References

- *Incorporating a TinCan LRS into an LMS*
- xAPI Launch Tool

LRS Credentials

When using packaged e-learning content, it's important to be aware that all content is downloaded to the learners' computers in addition to the JavaScript running any assessments and sending the data to the LRS/LMS is running on their computers.

This means whatever you do, ultimately, a tech-savvy learner (or a hacker who has compromised your organization's computers) could take control of the course and the tracking data. There's potential this type of user could access test answers, send false data, or pull back data from the LRS.

This risk applies to xAPI as much as to older e-learning standards, such as SCORM and AICC. It also potentially applies just as much to content launched from the LMS as to the zero security launchwrapper mentioned in the previous section. There are three potential responses you can take with this risk, each with pros and cons:

Launching xAPI content packages *(continued)*

OPTION 1: Accept the Risk.

This may sound shocking at first; but, in cases where the learners and assessments aren't high stakes, this may be the best option. It requires minimal work and doesn't prevent you from using the rich features of e-learning authoring tools that you're used to. Many organizations have been using SCORM for years without a single recorded case of a learner hacking the course package to fake his or her score.

However, most learning professionals have heard stories of executives getting their administrative assistants to complete courses for them or of teams emailing a cheatsheet of correct answers. There's easier ways to cheat at e-learning that don't require learners to be learning technologies interoperability standards experts. To put it another way: there's no point upgrading the locks on your doors if there's a great big hole in your wall.

OPTION 2: Restrict the scope of credentials used to limit the risk.

This option is a middle road. It doesn't remove the risk of someone accessing LRS credentials, but, by restricting the credentials' scope, you can limit the potential damage.

This option is really a spectrum of options with increasingly restricted credentials reducing the scope of any compromised credentials at the cost of increased work to implement. At a simple level, any created credentials should be configured with the LRS access setting to "isolated." This is the default value and restricts the credentials to only have access to data they stored. This means if credentials are compromised, the rest of the data in your LRS can't be accessed.

To further restrict credentials, you can decrease the scope across which each set of credentials is used. You can use a different set of credentials per course, or even use a different set of credentials each time a learner launches a course. For example, an LMS can be integrated with an LRS on the server side so it generates a new set of credentials each time a learner launches a course. These credentials can then be deleted via the same API after a configured number of minutes. So, if the credentials are compromised, they can only be used to access data about that session and only during a certain period of time before they're deleted.

This approach is more work to implement and still doesn't completely remove the risk. It does, however, have the benefit of reducing the potential risk consequences, while at the same time allowing you to continue to use the authoring tools you know and love. Talk to your LMS and LRS providers to see if such an integration is possible with their products.

Launching xAPI content packages *(continued)*

OPTION 3: Don't use packaged e-learning content; mark the assessments and track the learning interactions server side.

This option foregoes all packaged e-learning content in lieu of tracking and marking assessments on the server, rather than the learner's computer.

We include the marking of assessments here because there's little benefit sending the tracking data server side if a hacker can already interfere with the process before the tracking data is generated. Server-side tracking is not only more secure, but more reliable as you're not dependent on the learner's internet connection.

If you're using server-side tracking as a way to ensure learners don't cheat, remember what we said earlier about learners sharing answers with one another or having others take a course on their behalf. If you really want to make your assessments secure, you need to consider both the technological and social methods for getting around security checkpoints.

The downside of server-side tracking for content is that none of the major e-learning rapid authoring tools support it, heavily restricting your options in terms of creating content. Existing options for authoring server-side tracked content often lock you into that solution, preventing you from taking your content with you if you change tools.

Choosing the best approach for your organization is a balance with pros and cons of each approach; there's no one right answer. Instead, you'll need to consider the costs in effort and restrictions of implementing a more secure solution versus the likelihood and severity of the risks.

3) Check the Data

Take some time to explore your data for accuracy. This applies not only to xAPI statements you design internally, but also to statements created by others. If you haven't already, use the following xAPI Statement Review questions to make sure your statements are good.

Don't stop there, though. Use reports to probe at the data. If your reports look wrong, there's a good chance the underlying data is wrong.

PART 2: Let's Get Technical

Review the JSON statements.

As a first step in checking your data, check for syntax errors in the statements. Make sure the right properties have been used and check that statements are well populated with data. Look at the xAPI statements themselves (rather than reports at this stage) and use the questions in the following checklist to identify potential common mistakes:

General

- Does the overall stream of statements make sense and tell a whole story? Does the data show you what the learner did?
- Are statements only sent once (i.e., the same statement should not be sent multiple times with the same ID)?
- Is the timestamp different from the stored value? If they are the same, this indicates that the timestamp has been set by the LRS; in most cases it's more accurate for the application sending the data to set the timestamp. Is the timestamp accurate?
- Do statement properties contain any empty arrays or objects? If so, it's better not to include the property at all.
- Are the verb and object properly separated? That is, does the verb only describe the action and not the thing being acted on, and does the object only describe the thing being acted on and not the action taken?

Actor

- Is the actor.name property used? *(Note: If a person's name is planned to be brought in from another data source, such as an HR system, the person's name may not be required.)*
- Is the correct identifier used for the actor? See xAPI best practices above.
- Does the actor have the correct structure? (e.g., People may incorrectly populate an email address in the actor.account property or generate a fake email from an account.)

Verb

- Is the verb ID listed at https://registry.tincanapi.com/#home/verbs?
- Does the verb ID accurately represent the action? For innovative projects, an appropriate verb ID may not exist, but you can register an account on https://registry.tincanapi.com and submit a new verb ID.
- Are generic verb IDs avoided? (e.g., 'http://adlnet.gov/expapi/verbs/experienced')
- Is the verb display property populated?

PART 2: Let's Get Technical

Review the JSON statements. *(continued)*

Object

- Is the object activity ID correct? See xAPI best practices above.
- Is the object activity name populated? Does it clearly identify the activity so a user could easily chose it from a list?
- Is the object activity description populated with something sensible?
- Is the object activity type populated with an identifier from https://registry.tincanapi.com/#home/activityTypes? This is a common issue, but there's no reason not to populate the activity type in every statement.
- Are all properties of the object definition, including extensions, the same across all statements sharing the same activity ID?

  This is key because changes in values can affect reports; some reporting tools use the canonical definition of activities.

Result

- Is the result object appropriately used? This is optional, but all the result properties should be used wherever they are appropriate.
- Is result data sent multiple times? (e.g., sending the same duration in multiple statements can lead to double counting)
- Does the duration use a sensible structure? (e.g., P32.054S is preferable to P0D0H0M32.054S)

Context

- Is the context object used? Context should be included for all statements.
- Is context registration used? This should be used where possible and appropriate.
- Are context activities used? These should be used where appropriate.
- Are context activity definitions populated (see object above)? *Note: You may not need to populate a context activity definition if the data is sent in another statement.*
- Is a source context activity included? See *Finding the Source* for more information.

Review the JSON statements. *(continued)*

Extensions

Where extensions are used:

- Is there a non-extension property that can be used instead or in addition to? Often, first-time implementers use extensions in cases where they're unaware of a less popular xAPI statement property. Check the xAPI specification carefully.
- Has the extension been registered at https://registry.tincanapi.com/#home/extensions? Or, has an existing registry extension been used?
- Does the extension value match the expected value based on the registry description?
- Is the data structure of the extension value sensible? For example, composite values or JSON-encoded values are bad values because they're difficult to interpret. Unencoded objects/arrays and simple strings/numbers/booleans are good values because they're easy to interpret.
- Is the same data structure used every time the extension is used? For instance, if an extension contains an array in one statement, it should contain an array when used elsewhere.
- Is the extension in the right place? Extensions relating to the activity should be included in the activity definition and should always be the same value for every learner and every statement using that activity ID. Extensions relating to the overall experience belong in either the result or context, depending on if they relate to the context or result of the experience.
- Does the extension location match the expected location per the registry description (if specified)?

Note: *These questions are guidelines. There may be some situations where an expert might choose to deviate from the guidance implied by some of these questions.*

Test the data with reports.

The previous questions aren't exhaustive and even the best of us will miss things when reviewing a batch of raw JSON statements by sight. Once the statements pass the raw JSON review, move onto testing statements with reports. The reports and data used for testing should be as representative of the final reports and data as possible.

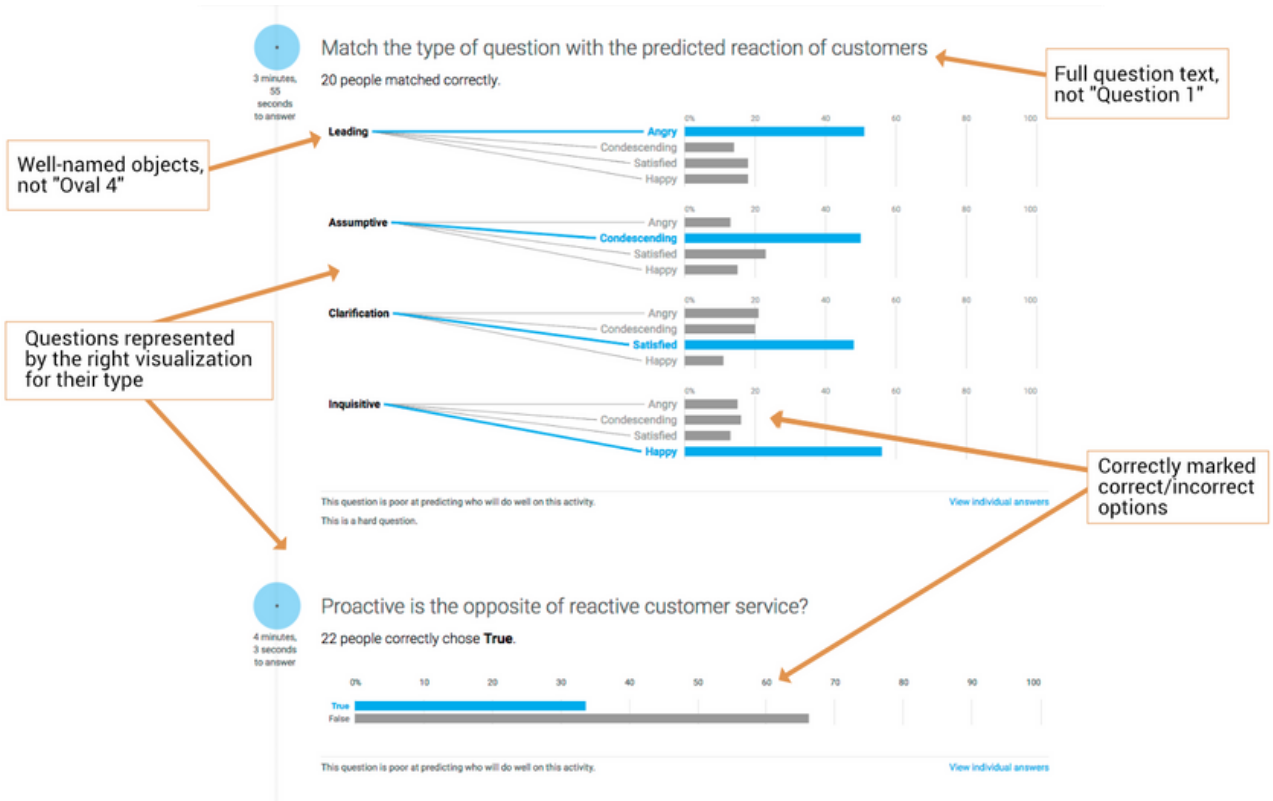
Test the data with reports. *(continued)*

As you look at reports, ask yourself if the data seems reliable. Does it make sense intuitively? Is the data clean and well structured? Do different data sources use similar xAPI statement structures, and is their data represented in the same way in reports? Are the metrics consistent with expectations? If possible, compare what you see to any existing reports. Do the reports from your LRS data give you the same results?

If your data doesn't line up with expectations and/or with existing reports, some possible reasons include:

- You're looking at a slightly different population of learners. *Are you filtering out inactive learners?*
- Measures and averages are being calculated slightly differently than previous reports.
- Some data isn't being reported, or is being reported multiple times due to a bug.
- Data is inaccurate due to a bug in translation.
- The original data from the data source is inaccurate.
- You've made an error configuring the reports. Learners are behaving in a way you didn't expect (i.e., The data is right. You're wrong.)

Take the time you need to figure out where and why the data doesn't match expectations and then adjust the data (or expectations) as needed. To give an idea of the process involved, the following graphic illustrates the kinds of checks you might make when reviewing statements from a typical e-learning course quiz using a report on that data.



About the Author

Andrew Downes
Learning & Operability
Watershed

With a background in instructional design and development, Andrew Downes creates learning platforms and experiences in academic and corporate environments. Now a learning and interoperability consultant with Watershed, Andrew is an expert in Evidence-Driven Learning and Learning Technologies Interoperability.

As an author and top contributor of xAPI (Experience API) and the majority of material on experienceapi.com, Andrew is a recognized xAPI expert who has delivered presentations, webinars, and training sessions across the globe.

www.watershedLRS.com