Diagnosing Content Package Issues

Prepared by Dave Hecker
Last updated 5/31/2017 by Dave Hecker

OVERVIEW

WHAT IS A CONTENT PACKAGE?

A “content package” is a ZIP file that contains files that display or launch content. Learner Community currently supports these kinds of content packages:

- AICC HACP
- SCORM 1.2
- SCORM 2004
- Web

To use a content package, usually it’s as simple as…

1. Upload it into Learner Community.
2. Create a Content Package learning activity and point to a valid “launch page” in the package.

…but sometimes you may encounter unexpected issues, especially if it’s the first time you’ve imported a content package from a new publisher or authoring tool.

HAVING TROUBLE?

When you encounter an issue with a content package, it is helpful to keep in mind that Learner Community is successfully launching many different content packages from many different clients, which means that the most likely source of any issue is the content package itself.

Although this document can’t anticipate the specific issue you’ve encountered, it does provide specific steps that you and/or your content vendor/publisher can use to diagnose issues. Once the source of an issue is identified and understood, it can usually be resolved (or at least worked around until a proper resolution can be made).

To diagnose an issue, start with the *Initial Checks* section that applies to any content package, and covers general issues such as content that fails to load. If the content is displaying properly but just isn’t communicating status or bookmarking information to Learner Community, then refer to the section for the specific type of content package.

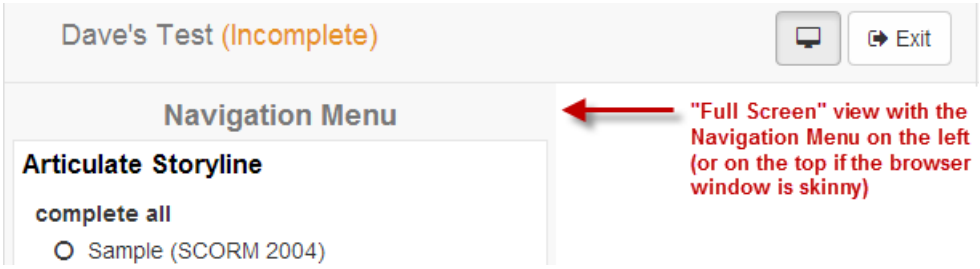
Some of the diagnostic steps require a web browser with good developer tools available. This document uses Google Chrome to walk through those diagnostic steps. If you’re already comfortable with the developer tools available in another web browser (such as Mozilla Firefox with the Firebug extension installed and enabled, Microsoft Internet Explorer 10 or higher, or other capable web browser) then you can use those tools (though you may still want to use Google Chrome to make it easier to follow the diagnostic steps).

INITIAL CHECKS

The following initial checks can be performed for all types of content packages. If additional diagnosis is needed, refer to the AICC or SCORM section for additional investigation instructions. *NOTE: There is not a special section for "Web" content packages, because they don't self-report status.*

FRAME VERSUS WINDOW?

If you are viewing the enrollment detail in the "Full Screen" view…



…then there is a quick initial test you can do. *(Skip ahead to the next section you're not using this view.)*

In the "Full Screen" view course content gets loaded into an iFrame that's embedded on the page. Although most modern authoring tools produce content that behaves properly in a frame, there may be some exceptions. For example, as of March 2014 Articulate Storyline on an iPad (and perhaps other tablets) doesn't display properly in an iFrame. Hopefully future releases fixed that exception, but if not you may need to provide a workaround.

To determine whether the issue you're seeing could be a "frames versus window" issue:

1. Click this button to "Leave Full Screen" view:

2. Launch the learning activity in the enrollment detail view that shows your portal's template. That view always opens a brand new web browser window for the content to operate in.

If the content package works properly when it is in its own dedicated browser window, but not when it's embedded in the full screen view's iFrame, then the content package is not "frames friendly." In this case, contact the package's author (or the publisher of the authoring tool you used to create the package) to see if they have an update that lets their content function properly in a frame.

If the publisher has no solution, then you might work around the issue by creating a different "start" page that would detect if it was running inside a frame and (if so) force a new window to open for the content.

BROWSER-SPECIFIC?

If the issue still occurs in the classic view, try testing the content in one or two different web browsers. If the issue happens in all web browsers, then it's probably not a browser-specific issue, so skip ahead to the next topic.

If the issue doesn't occur in some browsers, then it's likely a content-related issue that manifests only under certain web browsers, or certain web browser versions. For example, the last couple of releases of Internet Explorer introduced major changes, and some authoring tools had to be updated to address issues. Until the updates were available, enabling Internet Explorer's "compatibility view" mode for the URL where the course content resides would eliminate the issues. (If enabling "compatibility view" solves the issue, then it's an issue with the content.)

You should contact the authoring tool's vendor to determine if an updated version of the authoring tool is available. If not, search their forums and report the issue to see if there's a known resolution or workaround for the issue you're seeing.

MISSING FILES AND/OR UNUSUAL FILE TYPES?

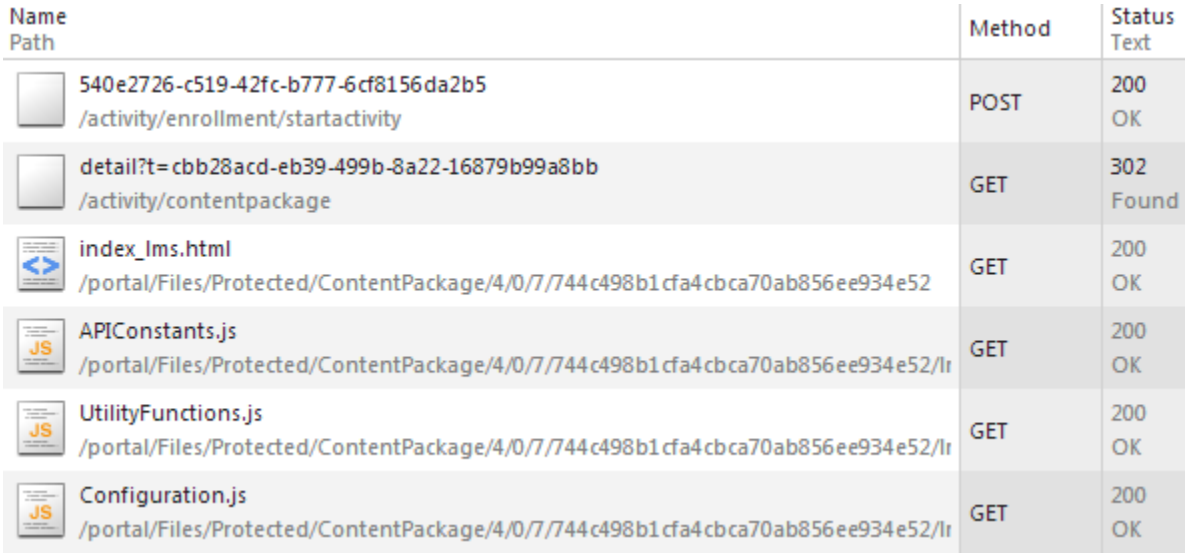
If some content isn't being displayed in any web browser, or the course gets "stuck" while loading and never starts playing, then check to see if the course is missing files and/or is using some uncommon file types.

Diagnosing in Google Chrome:

If you're not using the "Full Screen" view, click this icon to "Enter Full Screen" view:

Press **F12** to open the Developer Tools, and then click the **Network** tab that will show all network requests being made by the web browser.

Click the learning activity to launch the content package, and observe the activity on the Network tab. The first request should be a POST to /activity/enrollment/startactivity, which should followed by a series of additional requests that vary depending on the specific content package you launch. For example:



Name Path	Method	Status Text
540e2726-c519-42fc-b777-6cf8156da2b5 /activity/enrollment/startactivity	POST	200 OK
detail?t=cbb28acd-eb39-499b-8a22-16879b99a8bb /activity/contentpackage	GET	302 Found
index_lms.html /portal/Files/Protected/ContentPackage/4/0/7/744c498b1cfa4cbca70ab856ee934e52	GET	200 OK
APIConstants.js /portal/Files/Protected/ContentPackage/4/0/7/744c498b1cfa4cbca70ab856ee934e52/lr	GET	200 OK
UtilityFunctions.js /portal/Files/Protected/ContentPackage/4/0/7/744c498b1cfa4cbca70ab856ee934e52/lr	GET	200 OK
Configuration.js /portal/Files/Protected/ContentPackage/4/0/7/744c498b1cfa4cbca70ab856ee934e52/lr	GET	200 OK

The "Status" of each request should normally be "302 Found" or "200 OK". When you reach the point where the content gets "stuck" or does not display properly, scroll through the Network items to see if you see any GET requests which resulted in 404 Not Found:

GET	404 Not Found

A "404" error can mean that the requested file just doesn't exist in the content package, or it could exist but the web server might not be configured to serve that file type. (Most web servers are configured to serve common file extensions such as .HTM, .HTML, .TXT, .GIF, .PNG, .JPG, and so on. If the course content uses files that have uncommon file extensions, those requests might get a "404" error instead of receiving the actual file.

If you find any "404" errors:

1. Right-click anywhere on the row.
2. Choose "Copy Link Address"
3. Switch to an application where you can make diagnostic notes (e.g., Word or an email document) then Paste the link address.

If the link address ends with a common file type (e.g., .GIF or .JPG) then you can be reasonably confident that the web server is configured to serve those file types (especially if you can find other *successful* requests for the same file type) so that file is most likely missing from your content package. *NOTE: It's not uncommon for authoring tools to request a file that doesn't exist—one or two missing files, especially small placeholder graphics, are not likely the source of completely getting stuck, or having an entire page not display properly.*

After gathering your list of files that were not found, check the content package to be sure the file exists. You should inspect the actual content package version that's loaded into Learner Community by doing the following:

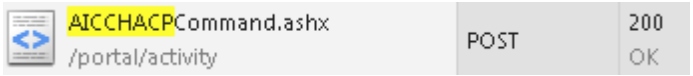
1. Go to the Admin Dashboard.
2. Open Products, and then select Learning Products.
3. Find the learning product you were testing, and then click Edit.
4. Click the Lesson Structure tab, find the learning activity you were testing, and then click Edit.
5. At the bottom of the "Edit Content Package Learning Activity" dialog, a complete outline is displayed of all the files that exist in the content package. Carefully follow the paths of each link that wasn't found.
    a. **If the file exists:**
        i. Carefully double-check the full path to be sure the full path matches exactly. If it does, then provide Learner Community support with the details (including the specific learning product, learning activity, and link address) to be investigated.
    b. **If the file doesn't exist:**
        i. Highlight and copy the "Package" name, and note which "Version" is currently selected.
        ii. Click "Change" and search for the package name to see if a newer version was uploaded, but just not selected. If so, consider selecting the newer package.
        iii. If the selected version is correct, then check your local ZIP file to see if the file exists there. If it's not there, then correct that issue. Once you add it (or if it's already there) then upload that as a new version of the existing package, and *then be sure to choose that new version* before you test again.

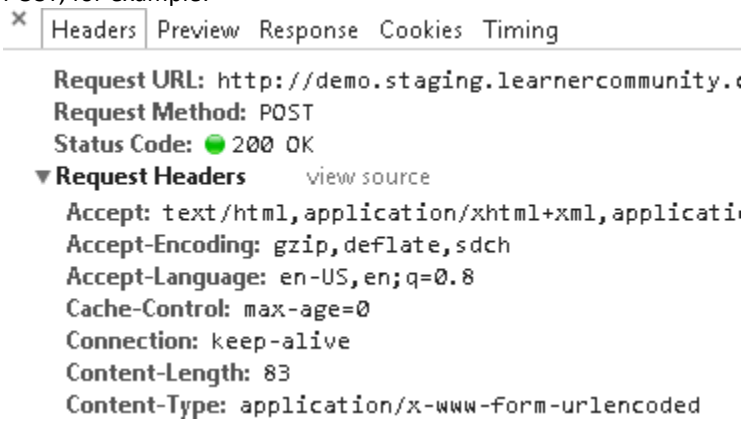
AICC CONTENT PACKAGES

Depending on the content page, you might be able to see AICC form POSTs happening to Learner Community. If so, those POSTs will also provide a view into the responses coming from Learner Community.

Here's how to try observing the course's AICC communication (using Google Chrome):

1. Launch the course (to view its Enrollment Detail page).
2. If you're not using the full screen view, click this icon to "Enter Full Screen" view:
   
3. Before launching the AICC learning activity, press F12 to open Developer Tools.
4. If the Developer Tools opens in the same window, click the "unlock into separate window" icon  to move the tools into their own window.
5. Click the Network tab. *NOTE: There may be network activity already displayed; click the "clear" icon* *to start fresh.*
6. Click the "filter" icon and in the text box type AICCHACP:
   

7. Launch the AICC learning activity, and review the Network activity. With the filter in place, you'll probably only see a few entries; these are the ones you want to inspect:
   

8. Click on "AICCHACPCommand.ashx" and the right side of the Developer Tools window will show details of that POST, for example:
   

INSPECTING EACH POST

By clicking each AICCHACPCommand.ashx POST in sequence, you can inspect each command issued by the course content, as well as the response that Learner Community provided to the course.

Data Submitted to Learner Community

On the **Headers** tab, find the **Form Data** section to view the data that the course posted in to Learner Community. The first POST usually looks something like this:

The session_id is different each time you launch a course. The “GetParam” command is the first call that content makes, which is a request to fetch current information for the activity. When making a “GetParam” POST the aicc_data parameter is empty.

Learner Community’s Response

Click the **Response** tab to view the data that Learner Community returned to the course. As mandated by the AICC specifications, each response will look something like this:

```
error=0
error_text=Successful
version=4.0
aicc_data=[core]
lesson_location=
lesson_status=not attempted
student_id=32a476c8d73145f080742af3b35c3368
student_name=Hecker, Dave
time=0000:19:59
credit=Credit
[core_vendor]
```

Each response will have the first 4 lines. Everything after `aicc_data=` is data that’s currently stored in Learner Community. As you interact with the course, the course will modify that data, and it will eventually POST that modified content back to Learner Community for storage.

SERVER-SIDE OBSERVATION

If you’re using content that uses a communication method that’s not seen by the web browser’s developer tools, there is a way to temporarily enable server-side diagnostics on the learner account that you’re using for testing (presumably on Staging). After diagnostics mode is enabled on a specific learner account, detailed logging of all server-side AICC communication will be captured for future analysis.

SCORM (1.2 AND 2004) CONTENT PACKAGES

DETERMINING VALID LAUNCH PAGES

1. Unzip the SCORM content package ZIP file on your local PC (be sure you're using the same ZIP file that's being used on the learning activity).
2. Open "imsmanifest.xml" into a text editor (e.g., Notepad, Notepad++, EditPlus, etc.) and search for the word **webcontent**—it usually exists once, but can exist many times. Each occurrence that's found should be on a line that starts with "<resource" followed by several attribute names and values.
3. In each "webcontent" resource you find, the **href** value is a valid "launch page." For example:

```
<resource identifier="__646XNvXCkGw_course_id_RES"
          type="webcontent"
          adlcp:scormType="sco"
          href="index_lms.html">
```

4. In the example, the valid launch page is "index_lms.html" *in the root of the package.* If the href value was "chapter1/indexlms.html" then the launch page "index_lms.html" would be found inside a "chapter1" folder.
5. Edit the learning activity to review the Launch Page that's currently selected. If it is not one of the valid launch pages identified in the imsmanifest.xml file, then browse through the content package to locate and select the proper launch page.

OBSERVING SCORM COMMUNICATION

When a SCORM-conformant LMS launches a SCORM course, it provides a standard Application Programmatic Interface (API) that the course content can locate and communicate with. Typically the LMS does nothing more than load the course content—it then just lies dormant, waiting for the SCORM course to issue commands.

While some courses don't contact the LMS immediately, most courses do. Most courses will immediately attempt to locate the SCORM API that's provided by the LMS, and then issue the "initialize" command to inform the LMS that a SCORM session is starting. Assuming that initialization is successful, the course content will usually issue additional commands (such as get a value; set a value; permanently save progress changes to the LMS).

Both the course content and the SCORM API code exists in the learner's web browser, which makes it possible for you to observe exactly what any course is telling Learner Community's SCORM API to do. In fact, Learner Community's SCORM API code was specifically designed to expose a clear view into that activity to help "demystify" SCORM, and to give you and your content providers a great diagnostic tool.

Here's how to observe SCORM communication (using Google Chrome):

1. Launch the course (to view its Enrollment Detail page). *NOTE: The following works whether you're in "Full Screen View" or not (either view is fine).*
2. Before launching the SCORM learning activity, press F12 to open the Developer Tools.
3. If the Developer Tools opens in the same window, click the "unlock into separate window" icon  to move the tools into their own window.
4. Click the Console tab. *NOTE: There may be messages already displayed in the console. You can click the "clear console log" icon*  *to start fresh.*
5. In the Console window's prompt, type *exactly* the following case-sensitive command:

   scormNS.doLog(true)

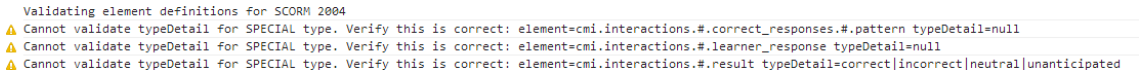
   then press Enter. You should see the following:

```
> scormNS.doLog(true)
  "Logging Enabled"
```

6. Now launch the SCORM learning activity, and watch the Console window to see what (if any) communication the SCORM course is making to Learner Community's SCORM API. *TIP: Position the "Developer Tools" window, and the browser window that has the SCORM course content so that you can see both windows at the same time.*

FIRST LAUNCH

The first time a SCORM 1.2 or 2004 course is launched, Learner Community will prepare its API—the console should show a message such as "Validating element definitions for SCORM X" where X is either 1.2 or 2004. That validation does internal sanity checks on some data definitions. Depending on the SCORM version you may see some warnings about certain items that aren't validated automatically. For example, here's the first entries when launching any SCORM 2004 content package:



Those are just expected FYI warnings, not an error. If you launch another SCORM 2004 activity during the same session, the element definitions will already be validated, so those messages would not appear. (If you launched a SCORM 1.2 activity during the same session, those element definitions would be validated once.)

After validating element definitions, Learner Community essentially goes dormant and just waits for the SCORM course to locate the API and issue valid commands. If no other console activity appears, then the SCORM course may be one that doesn't do any communication until you exit the course, or maybe not even until you complete the entire course in a single session. However, if it **never** initiates communication, then it may not be a SCORM course; perhaps it is a SCORM course that can't locate the SCORM API provided by Learner Community; perhaps the incorrect "launch page" is selected. Check with your content publisher to resolve any of those issues.

Most courses will immediately attempt to locate the SCORM API provided by the LMS, and will alert the learner if the API could not be located. *NOTE: Learner Community's SCORM API is located "in plain sight" at the root of the enrollment detail window, so hopefully you'll never encounter a course that's unable to find it!*

UNDERSTANDING THE MESSAGES

After locating the SCORM API, most courses will immediately issue the command that starts up the API. The specific command varies by SCORM version: in SCORM 1.2 the command is `LMSInitialize`, while in SCORM 2004 the command is `Initialize`. Here's an example of the first launch of a SCORM 2004 content package:

```
/ SCO called Initialize("")
|   Fetching data from LMS.
|   Fetch succeeded.
|   ScormDatabase - 5 keyValuePairs:
|       cmi.entry = ab-initio
|       cmi.mode = normal
|       cmi.total_time = P0Y0M0DT0H0M0S
|       cmi.completion_status = unknown
|       cmi.success_status = unknown
\ Result for Initialize("") (OK) -> "true"
```

Each time the course content issues a valid SCORM command, Learner Community's SCORM API will output a similar "block" of messages to show you exactly what's happening. The command starts with "`/ SCO called`" followed by the command name and parameters issued by the course content ("SCO" is short for "Shareable Content Object" which is a fancy phrase which you can think of as "the course content").

The command ends with "`\ Result for`" again followed by the command name and parameters issued by the SCO (the course content) followed by "`(resultStatus) -> returnValue`. In the example above, the resultStatus is OK (meaning no error) and the value that the command is returning to the SCO is the text "true". If a SCO issues a command that isn't appropriate or valid at the time, the resultStatus would not be OK but instead would show "`Error #=Info`" where `#` is an error number as defined in the SCORM specifications, and `Info` is a description of that error code, possibly followed with additional details.

In between the command start and end lines might be additional details that start with "`|`". In the example above, Learner Community's SCORM API tells you exactly what it was doing:

- `Fetching data from LMS.`
  Learner Community's client-side API is requesting information from a Learner Community server.
- `Fetch succeeded.`
  In this case, the fetch was successful. If any error had occurred it would be noted here instead.
- `ScormDatabase - 5 keyValuePairs` (followed by the 5 "`key = value`" pairs).
  This lists all the values currently defined in the local "ScormDatabase" that the client-side API is maintaining in memory in your web browser. Those values will change as the SCO interacts with the API, and certain commands will again display the current local ScormDatabase values (for example, the "save changes" command will output the values so you can see exactly what values are being submitted to the LMS).

NOTE: Other sources could also add messages into the Console that are completely unrelated to the SCORM API. In general, if the message is outside a group of messages as shown above, or appears within the group but doesn't follow the same display pattern, then it's not a message from Learner Community's SCORM API.

More Examples

After starting up the API, a typical course will issue one or more commands to "get" or "set" certain values that it needs. After issuing certain commands, SCOs will often call a "get last error" command to determine whether the previously-issued command was successful. Continuing the example of the first launch of a SCORM 2004 content package, we'll walk through what happened after the Initialize command executed successfully:

`/ SCO called GetValue("cmi.success_status")`
`\ Result for GetValue("cmi.success_status") (OK) -> "unknown"`
 	
`/ SCO called GetLastError()`
`\ Result for GetLastError() (OK) -> "0"`
	SCO retrieved the current "**success** status" value, then verified the retrieval was successful.

`/ SCO called GetValue("cmi.completion_status")`
`\ Result for GetValue("cmi.completion_status") (OK) -> "unknown"`
 	
`/ SCO called GetLastError()`
`\ Result for GetLastError() (OK) -> "0"`
	SCO retrieved the current "**completion** status" value, then verified the retrieval was successful.

`/ SCO called SetValue("cmi.completion_status", "incomplete")` `\ Result for SetValue("cmi.completion_status", "incomplete") (OK) -> "true"`	SCO set the "**completion** status" to "incomplete".
`/ SCO called SetValue("cmi.exit", "suspend")` `\ Result for SetValue("cmi.exit", "suspend") (OK) -> "true"`	SCO set the "exit" value to "suspend".

`/ SCO called GetValue("cmi.mode")`
`\ Result for GetValue("cmi.mode") (OK) -> "normal"`
 	
`/ SCO called GetLastError()`
`\ Result for GetLastError() (OK) -> "0"`
	SCO retrieved the "mode" value, then verified the retrieval was successful.

`/ SCO called SetValue("cmi.session_time", "PT0.17S")` `\ Result for SetValue("cmi.session_time", "PT0.17S") (OK) -> "true"`	SCO set the "session time" value to 0.17 seconds.

`/ SCO called Commit("")`
`|  ScormDatabase - 7 keyValuePairs:`
`|  cmi.entry = ab-initio`
`|  cmi.mode = normal`
`|  cmi.total_time = P0Y0M0DT0H0M0S`
`|  cmi.completion_status = incomplete`
`|  cmi.success_status = unknown`
`|  cmi.exit = suspend`
`|  cmi.session_time = PT0.17S`
`|  Submitting changes to LMS...`
`|  Submission succeeded.`
`\ Result for Commit("") (OK) -> "true"`
	Now that the SCO has set some values, it decided to issue the Commit command, which asks the LMS to store all the current values in permanent storage.
	
	The LC SCORM API lists all the current local values, then submits to the LMS, and finally reports that the submission succeeded.
	
	At this point if the learner lost their internet connection or abruptly killed their web browser window, whatever progress had been saved by the course content would be the starting point the next time the learner launched the same learning activity.

Sanity Checks

Learner Community's SCORM API performs basic "sanity checks" on the commands that the SCO issues. If the SCO provides data that can be identified as not being valid for a particular command, it will be reported. For example, here's what would be reported if a SCO tried setting the "completion status" to an invalid value of "bogus":

```
/ SCO called SetValue("cmi.completion_status", "bogus")
\ Result for SetValue("cmi.completion_status", "bogus")
   (Error 406=Data model element type mismatch./Element
   "cmi.completion_status" cannot be set to the value "bogus" because it
   does not fit the element data type.) -> "false"
```

Instead of `(OK) -> "true"` the result status is reporting error 406 (data model element type mismatch) with the additional detail that *"cmi.completion_status" cannot be set to the value "bogus" because it does not fit the element data type*.

Whether a particular error is "meaningful" (that is, whether it is an indication of a problem with the way the course content is written) or "expected" can depend on the context. In general, errors from a "get value" command are usually not a problem, while errors from a "set value" usually indicate an error where the course content is not following the SCORM rules. Rather than accepting a value that doesn't adhere to the documented SCORM standards, the improper value will be ignored.

Final Example

To "round out" the example of the first launch of a SCORM 2004 content package, we completed the course, and are navigating to a different learning activity. Typically the SCO will issue a "commit" to be sure that all the current data values, and will then issue the command that shuts down the API; here's the final command the example course issued:

```
/ SCO called Terminate("")
|   ScormDatabase - 8 keyValuePairs:
|      cmi.entry = resume
|      cmi.mode = normal
|      cmi.total_time = 0000:00:10.37
|      cmi.completion_status = completed
|      cmi.success_status = passed
|      cmi.exit = suspend
|      cmi.session_time = P0Y0M0DT0H0M0S
|      cmi.suspend_data = 2E18607040509810013110000~2f1~2b1c
|   Submitting changes to LMS...
|   Submission succeeded.
\ Result for Terminate("") (OK) -> "true"
```

Learner Community's SCORM API listed all the current local values, submitted them to the LMS, and the submission succeeded. *NOTE: The value for "suspend_data" is truncated in the image above, but the full value is displayed in the console.*

Many of the SCORM data element names and values are self-explanatory (e.g. "cmi.completion_status" indicates whether the SCO is complete or not). Some data elements require specific values, while others (such as "cmi.suspend_data") accept almost any value provided by the SCO—and the values themselves only mean something to the SCO itself. (Refer to SCORM documentation for details about a data element.)

SERVER-SIDE OBSERVATION

If your SCO misbehaves **only** on a device that doesn't have an easy "Developer Tools" option (for example, an iPad) then there is a way for Leaner Community support to temporarily enable server-side diagnostics on the learner account that you're using for testing (presumably on Staging). After diagnostics mode is enabled on a specific learner account, all server-side SCORM communication will be logged for future analysis.

Server-side logging can't show the same level of detail that the client-side messages can, because most SCORM API commands execute entirely on the client-side. Only *initialize*, *commit*, and *terminate*/*finish* commands cause the client-side API to communicate with the server—and there are actually only two server-side commands:

- BeginSession (triggered when the SCO issues the *initialize* command)
- UpdateSession (triggered when the SCO issues either a *commit* or a *terminate*/*finish* command).

You can still learn a lot from the less granular server-side logging enabled. After testing the content:

> **If the server-side log was empty,** then you would know that *either* the content didn't issue any valid SCORM API commands on the device in question, *or* it did but LC's SCORM API was somehow failing on that device (which is not likely—and is definitely not the case if you have any other SCORM content that functions properly on the same device).
>
> **If the server-side log shows activity,** then you would know that not only is the content successfully issuing SCORM API commands on the device, but also that LC's SCORM API is processing those commands successfully. (If one command functions successfully, all commands should function successfully—as long as the content is issuing valid SCORM API commands.) The log will also identify which command was executed (BeginSession or UpdateSession) along with any values received.