



Smarter Balanced Digital Library
Exemplar Instructional Module Technical Documentation

Smarter Balanced Digital Library (RFP 23)
Exemplar Instructional Module
Technical Documentation

Prepared for:



by:

Amplify.



Smarter Balanced Digital Library Exemplar Instructional Module Technical Documentation

[Revision History](#)

[Review History](#)

[Document Maintenance](#)

[Acronyms List](#)

[Introduction](#)

[1.1 Purpose](#)

[1.2 Supplementary Resources](#)

[1.3 Example Module](#)

[Design Overview](#)

[2.1 Module Blueprints](#)

[3. General Module Technical Overview](#)

[3.1 Key Software Components](#)

[3.2 Module Requirements](#)

[3.2.1 Module Browser Requirements](#)

[3.2.2 Module Accessibility Requirements](#)

[3.2.3 Module Aspect Ratio Requirements](#)

[3.2.4 Module Upload Requirements](#)

[4. Module Delivery](#)

[4.1 Module Upload to the Digital Library](#)

[5. Example Module Technical Overview](#)

[5.1 Example Zip Contents](#)

[5.2 End User Requirements](#)

[5.3 Navigational Controls](#)

[5.4 Configuration Script](#)

[5.5 Available runJS JavaScript Functions](#)

[5.6 Module Review Feedback Mechanism](#)

[5.7 Amplify Employed Content Production Techniques](#)

Revision History

Version	Date	Author(s)	Description
1.0	2014 08 10	Frank Walsh	Template draft
1.1	2014 08 20	Frank Walsh	Content Authorship
1.2	2014 09 10	Frank Walsh	Content Updates
2.0	2014 10 01	Frank Walsh	Major Content Revision
2.1	2014 10 29	Frank Walsh	Major Content Revision

Review History

Date	Reviewer(s)	Description
2014 08 25	Frank Walsh	Content and Technical Review
2014 09 10	Frank Walsh	Content and Technical Review
2014 09 12	Stanley Chung	Content Review
2014 10 06	Stanley Chung	Content Review

2014 10 21	Linda Payson	Content Review
------------	--------------	----------------

Document Maintenance

This document will be updated as needed, as the project proceeds through each phase of the system development life cycle (SDLC). This document contains a revision history log located directly after the table of contents. When changes occur, the document’s revision history log will reflect an updated version number as well as the date, the owner(s) making the change(s), and a description of the change(s). The version number will follow the following guidelines:

Table 2: Version Change Guidelines

Change Type	Version Update (“#” reflects change)	Description
Major	#.1	Major changes will be recorded when this document is used for a different deliverable (i.e. LRN Conceptual Design Part 1 and Part 2).
Minor	1.#	Minor changes will be recorded when sections are significantly changed or completed.

Acronyms List

Table 2: Acronyms List

Acronym	Definition
API	Application Program Interface



Smarter Balanced Digital Library Exemplar Instructional Module Technical Documentation

AWS	Amazon Web Services
CDN	Content Delivery Network
CMS	Content Management System
CORS	Cross-origin resource sharing
CSS	Cascading Style Sheet
DL	Digital Library
GCS	Google Cloud Storage
GIF	Graphic Interchange Format
HTML5	Hyper Text Markup Language 5
JAWS	Job Access With Speech Screen Reader
JPEG	Joint Photographic Experts Group Standard
JS	JavaScript
JSON	JavaScript Object Notation
LCMS	Learning Content Management System
LMS	Learning Management System
MPEG	Moving Picture Experts Group
NVDA	NonVisual Desktop Access Screen Reader
OGG	OGG Container Format



Smarter Balanced Digital Library Exemplar Instructional Module Technical Documentation

OGV	OGG Video Container Format
OS	Operating System
PNG	Portable Network Graphics
PX	Pixels
REST	Representational State Transfer
SCORM	Sharable Content Object Reference Model
SDLC	Software Development Life Cycle
SRT	SubRip Caption Format
UI	User Interface
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines
WOFF	Web Open Font Format
xAPI	Experience Application Program interface

1. Introduction

1.1 Purpose

This document outlines the infrastructure, technology and configuration used to build a Digital Library Exemplar Instructional Module. This document also serves as a summary of best practices and overview of how individuals might author modules to match the design and functional capabilities of an Amplify authored Exemplar Instructional module.

1.2 Intended Audience

The intended audience for this document are individuals with experience developing web content using HTML, CSS and JavaScript as described in Section 3.1 of this document.

1.3 Supplementary Resources

The following resources accompany this documents and should be consulted when appropriate:

Document Number	Filename	Description
1	ExampleModule.zip	Example module and mock Digital Library Launch page, Adobe Video Encoding Presets, a Sample Adobe Captivate Project File, and Layered Adobe PSD art files used in module thumbnail and cover profile page image production.
2	Exemplar-Module-Blueprint.pdf	Blueprint providing overview of the components of an Amplify Authored Exemplar Module.
3	Score-Reports-Module-Blueprint.pdf	Blueprint providing overview of the components of an Amplify Authored Score Reports Module.

Exemplar Instructional Module Technical Documentation

4	Assessment-Literacy-Module-Blueprint.pdf	Blueprint providing overview of the components of an Amplify Authored Assessment Literacy Module.
5	Module_Functionality_Specification s.docx	Functional Specification Overview For Amplify Authored Modules.

1.4 Example Module Overview

The ExampleModule.zip noted in Section 1.3 shows an example of a module that was authored by Amplify and includes both a mock Module Launch page and an example module. It uses an Amplify specific user interface, however this interface is not required to upload modules to the Digital Library. The technical upload requirements are specified in Section 3.2.4 of this document..

2. Design Overview

As a basis for design, Amplify developed a module approach which aligned to the specifications outlined in RFP-23, including expectations that exemplar modules include:

- Multiple embedded video and text examples of the instructional shiftsA highly interactive user experience
- Multiple places to click for deeper exploration
- Video vantage points from different stakeholder groups
- Short focused-reading excerpts
- Printable takeaways
- Evidence collection tools & follow-up planning materials
- Be accessible to users who have viewing disabilities
- Be portable (self contained) and potentially utilized on other LMS and CMS platforms

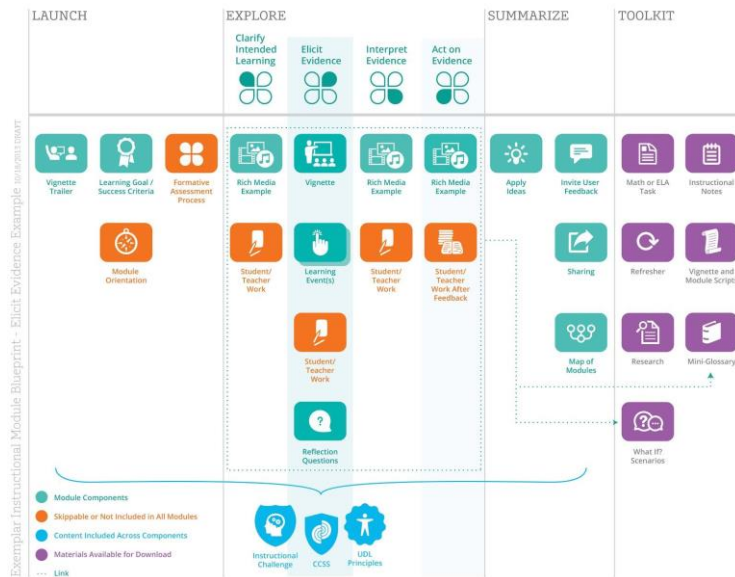
2.1 Module Blueprints

Initial module requirements are defined in three blueprints which include an overview of each module type’s structure and interactivity. These blueprints provide a visual overview of a

module’s components in the context of the Formative Assessment Process and Common Core State Standards.

The blueprints are named Exemplar Module Blueprint, Score Report Module Blueprint, and Assessment Literacy Module Blueprint. They are represented in Section 1.3 as Documents 2, 3 and 4.

Additionally, the Module Functionality Specifications document, (Section 1.3 Document 5) provides a summary of the functional definition of each element referenced in the module blueprints.



Screenshot of Exemplar Module Blueprint - This blueprint illustrates the basic organization of a module and various possible configurations and components:

3. General Module Technical Overview

Important capabilities of a module include:

- The ability to be wrapped inside a variety of eLearning packaging standards (e.g. Common Cartridge, SCORM, TinCan xAPI)

- A highly compatible interface which is both tablet and desktop pc friendly (e.g. No Flash & Responsive Design)
- An accessible interface for users with visual, hearing, motor and cognitive disabilities (i.e. WCAG 2.0 and 508).

As with most CMS systems, the Content display, management and storage capabilities of the Digital Library are predominantly driven to provide users access and collaboration capabilities focused on discrete content objects (e.g. Downloadable PDF, Word, Excel Document). Consequently, the Digital Library does not maintain learner course and activity history.

A module is expected to be a self-contained HTML5 Microsite (<http://en.wikipedia.org/wiki/Microsite>) providing consistent, accessible and embedded navigational controls and print/email functionality allowing a user to self-record their personalized learning outcomes without any connectivity to a learning record store to record progress.

3.1 Key Software Components

A modules internal structure is expected to consist of these standard web development components:

HTML5 - HTML5 is a core technology markup language of the Internet used for structuring and presenting content for the World Wide Web. It is the fifth revision of the HTML standard (created in 1990 and standardized as HTML 4 as of 1997) and, as of December 2012, is a candidate recommendation of the World Wide Web Consortium (W3C). Its core aims have been to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices (web browsers, parsers, etc.). HTML5 is intended to subsume not only HTML 4, but also XHTML 1 and DOM Level 2 HTML.

JavaScript - JavaScript (JS) is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content displayed.

CSS - Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a markup language. CSS is a cornerstone specification of the web and almost all web pages use CSS style sheets to describe their presentation. CSS is designed primarily to enable the separation of document content from document presentation, including elements such as the layout, colors, and fonts. This separation can improve content

accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple pages to share formatting, and reduce complexity and repetition in the structural content (such as by allowing for tableless web design).

Web Open Font Format - The Web Open Font Format (WOFF) is a font format for use in web pages. WOFF is essentially OpenType or TrueType with compression and additional metadata. The goal is to support font distribution from a server to a client over a network with bandwidth constraints.

PNG and GIF images - Portable Network Graphics (PNG, pronounced "ping"), is a raster graphics file format that supports lossless data compression. The Graphics Interchange Format (better known by its acronym GIF;) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.

MPEG-4 Part 14 Encoded Video - MPEG-4 Part 14 or MP4 is a digital multimedia format most commonly used to store video and audio, but can also be used to store other data such as subtitles and still images. Like most modern container formats, it allows streaming over the Internet.

OGG Encoded Video - Ogg is a free, open container format maintained by the Xiph.Org Foundation. The creators of the Ogg format state that it is unrestricted by software patents and is designed to provide for efficient streaming and manipulation of high quality digital multimedia. OGG is included to provide video playback functionality to users of Mozilla Firefox.

3.2 Module Requirements

3.2.1 Module Browser Requirements

Minimum requirements are aligned with those of the Digital Library platform:

Use of a Modern Browser - Modules are to make use of HTML5 standards for playing, scrubbing and replaying HTML5 video and CSS interface styling (e.g. rounded corners, partially opacity, web fonts). To support access to modules using these features, an end user must have a minimum browser version of:

- Internet Explorer 9.0
- Mozilla 3.0
- Safari 3.0
- Chrome 27

- Opera 9.5

Note the Digital Library is designed to detect if a user attempts to launch modules from an unsupported browser and will notify the user via pop-up window.

No Required Browser Plugins - Smarter Balanced requires that modules be constructed utilizing authorship techniques that do not make use of proprietary browser plugins such as but not limited to:

- Flash
- Microsoft Silverlight
- JAVA

3.2.2 Module Accessibility Requirements

Modules are required to comply with both Section 508 of the Rehabilitation Act and the Web Content Accessibility Guidelines (WCAG) 2.0 Level A criteria published by the W3C (World Wide Web Consortium) available here: <http://www.w3.org/TR/WCAG20/>

Accessibility should consider four major disability categories:

1. Users with Visual Disabilities (e.g. Blindness, Color-blindness, Low Vision)
2. Users with Hearing Disabilities (e.g. Deafness and hard-of-hearing)
3. Users with Motor Disabilities (e.g. Slow response time, limited fine motor control, difficult using a keyboard/mouse)
4. Users with Learning Disabilities (e.g. Distractibility, Inability to remember or focus on large amounts of information)

Accessibility should seek to provide “access to and use of information and data that is comparable to the access to and use of information and data by those who are not individuals with disabilities” as defined by Section 508 of the Rehabilitation Act.

Furthermore, the four principles of WCAG 2.0 should serve as directives when authoring a module. Modules should be:

- **Perceivable** - Available to the senses (vision and hearing primarily) either through the browser or assistive technologies (e.g. screen readers, screen enlargers, etc.)

Exemplar Instructional Module Technical Documentation

- **Operable:** Users can interact with all controls and interactive elements using either the mouse, keyboard, or an assistive device
- **Understandable:** Content is clear and limits confusion and ambiguity
- **Robust:** A wide range of technologies (including old and new user agents and assistive technologies) can access the content

Principles	Guidelines	Level A
1. Perceivable	1.1 Text Alternatives	1.1.1
	1.2 Time-based Media	1.2.1 – 1.2.3
	1.3 Adaptable	1.3.1 – 1.3.3
	1.4 Distinguishable	1.4.1 – 1.4.2
2. Operable	2.1 Keyboard Accessible	2.1.1 – 2.1.2
	2.2 Enough Time	2.2.1 – 2.2.2
	2.3 Seizures	2.3.1
	2.4 Navigable	2.4.1 – 2.4.4
3. Understandable	3.1 Readable	3.1.1
	3.2 Predictable	3.2.1 – 3.2.2
	3.3 Input Assistance	3.3.1 – 3.3.2
4. Robust	4.1 Compatible	4.1.1 – 4.1.2

Accessibility testing should be performed using a combination of Automated Validation Testing Tools, Quality Assurance/User Testing (using Screen Readers and Accessibility Configured Browsers) and the standards defined by the Website Accessibility Conformance Evaluation Methodology provided by WCAG (<http://www.w3.org/TR/WCAG-EM/>).

Amplify module testing has been preformed utilizing SortSite (<http://www.powermapper.com/products/sortsite/>) and Quality Assurance/User Testing using Freedom Scientific’s JAWS screen reader (<http://www.freedomscientific.com/Products/Blindness/Jaws>), NVDA screen reader (<http://www.nvaccess.org/>) and Mozilla Firefox Fangs Browser extension (<https://addons.mozilla.org/en-US/firefox/addon/fangs-screen-reader-emulator/>).

Exemplar Instructional Module Technical Documentation

3.2.3 Module Aspect Ratio Requirements

The Digital Library is designed to launch modules in a 1600 x 900 pixel frame which launches on top of the Digital Library module launch screen. The frameScale.html file referenced in Section 3.2.4 of this document and included in the example module package provides responsive scaling for other screen sizes while ensuring a 16:9 aspect ratio is maintained in a module.

3.2.4 Module Upload Requirements

Module Size - The Digital Library requires that zipped modules be no greater than 250MB for upload to the Digital Library. This requirement exists to ensure Smarter Balanced predictable & affordable CDN delivery charges, module portability, and a dependable user experience when both uploading (as an admin) and accessing (as an end user) a module.

Module Folder Structure - While details regarding the Amplify/Smarter Balanced module user interface design and capability are provided in Section 5 of this document, only two major requirements exist when uploading a content module to the Digital Library:

1. The first folder level inside a zip should contain a single folder which is named to match the title of a module (this title is displayed in a bar displayed beneath the Digital Library Cover Profile Launch Pane for a module):

e.g.

ZIP: MODULE.ZIP

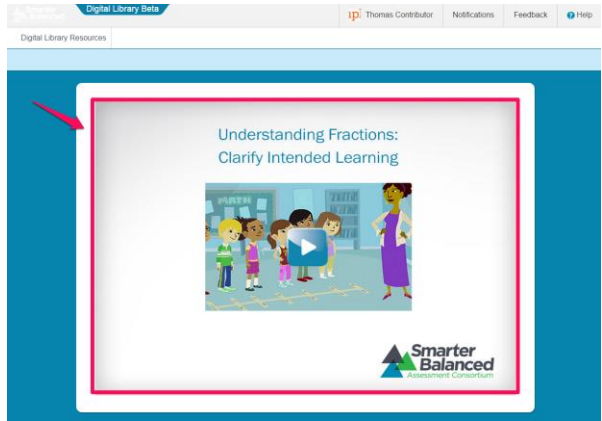
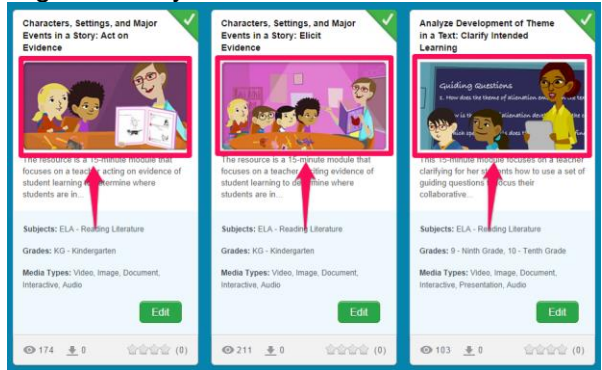
---CONTENTS OF MODULE.ZIP: ---

FOLDER : "Module Name"

--- CONTENTS OF "Module Name" which comprise module's microsite ---

2. Inside the "Module Name" folder the Digital Library confirms on module upload that a sub-folder exists named "**dlcomponents**" containing the following files:

Filename	File Description
CoverPage.html	This file is referenced by the Digital Library to provide a large preview pane for the module

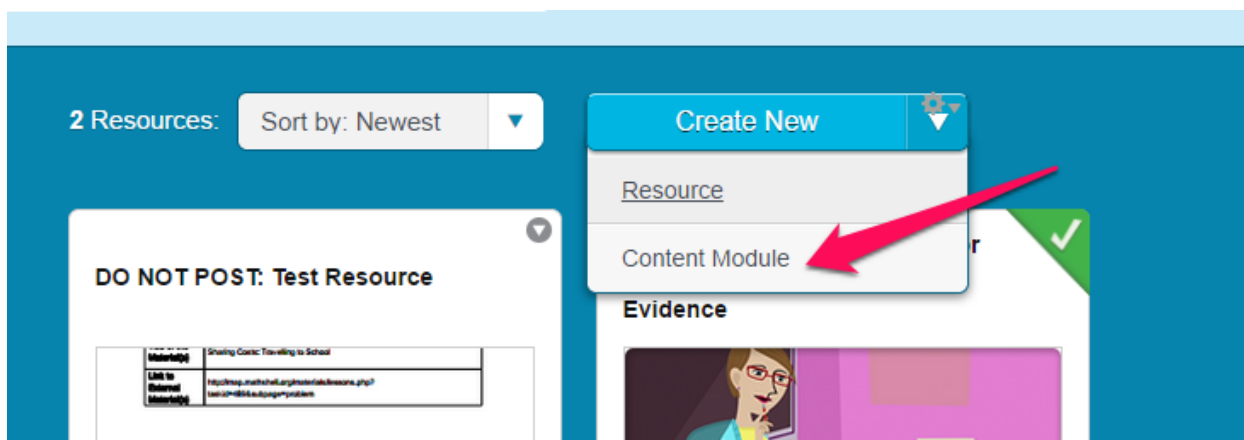
	<p>on the Digital Library item cover profile page. (600 px height x 850 px width). The CoverPage.html located in the attached example module provides javascript which scales this view to varying screen sizes.</p> <p><i>e.g. Screenshot shows red square highlighting purpose of CoverPage.html in the Digital Library Cover Profile Page.</i></p> 
<p>thumbnail.png</p>	<p>This file is referenced by the Digital Library when a user performs a search.</p> <p>138 px height by 268 px width</p> <p><i>e.g. Screenshot below includes red squares showing the purpose of thumbnail.png in Digital Library</i></p> 

<p>frameScale.html</p>	<p>This is the HTML file the Digital Library references when a user clicks to launch a module. This provides a means to resize modules to varying screen sizes without code dedicated to performing responsiveness scaling internal to your module's index page.</p> <p>Note: In our example module, frameScale.html contains an auto-scaling <iframe> which references an index.html file located in the root of the example module. However, there is no requirement stipulating where the <iframe> in this HTML file points. Only that frameScale.html exist as it is the starting point a user's browser is directed to when launching a module.</p>
<p>DLCover.jpg (<i>optional</i>)</p>	<p>In our example module, CoverPage.html references DLCover.jpg to provide the graphics used in the module launch page. No functional requirement exists requiring this file or this filename but the current CoverPage.html does reference it, and will appear blank if not included. Alternatively, CoverPage.html could be modified to point to any other file or contain embedded SVG graphic.</p> <p>NOTE: CoverPage.html does not support interactive content. While it could contain content that visually changes or moves, there is no support for interactivity by an end user clicking on different locations inside Cover Profile Page launch square. Clicking on any location in the Cover Profile launch square would instead launch the module.</p>

4. Module Delivery

4.1 Module Upload to the Digital Library

You should have access to the My Resource tab. From the Create New menu option select content module. If you do not have access the My Resource tab or the Content Module option contact a Digital Library system administrator who will grant you the permissions necessary to post a module.



During the upload process, a user selects a zipped module and then attaches it to a cover profile:

- Step 1 - Log in to the Digital Library
- Step 2 - Click the **My Resources** tab from the top navigation
- Step 3 - Click the drop down beside **Create New** then click **Content Module**
- Step 4 - Click **Continue** to agree to the terms of Creating a Resource
- Step 5 - Fill out the General Tab of the Module Cover Profile
- Step 6 - Click **Continue** to move to the **Materials** tab of the module
- Step 7 - Click **Choose File** select your module zip and then Press **Upload**.
- Step 8 - Complete the remaining Sections of the Cover Profile using **Continue** to move through the process.

Exemplar Instructional Module Technical Documentation

Add Content Module *

This is the content module with which you want users to interact. You can add a HTML5 captivate package file with a zip extension.

Content Module

ela03.zip

Add Secondary Module Materials

These will be available in addition to module content. You can add a combination of videos, images and documents to

After a module is uploaded and the cover profile completed, it will become accessible to end users within 10 minutes.

You can check to make sure the resource is available by going to the Digital Library Resource tab and filtering by selecting the module type (Assessment Literacy, ELA Exemplar Instructional, Math Exemplar Instructional or Score Report modules) to find the uploaded module.

5. Example Module Technical Overview

This section walks through a sample module which can be used as a template for building future modules.

5.1 Example Zip Contents

TABLE 1: Contents of ZIP Root Folder:

TYPE	NAME	DESCRIPTION
FOLDER	DLModulesFolder	Folder which simulates the global module folder where all modules are stored on the Digital Library. It contains the example module folder Inside this folder, the Amplify example module may be found in the

Exemplar Instructional Module Technical Documentation

		<p>“ExampleModule” folder titled “ExampleModule”</p> <p>Contents of this folder listed in TABLE 2 below.</p>
FOLDER	example_files	Folder containing all necessary images, stylesheets and script files used to simulate the Digital Library Cover Profile Module Page (index.html).
FOLDER	fancybox	Folder containing all files used to simulate the module launching in a lightbox on top of the Cover Profile Module Page.
HTML	index.html	HTML page which simulates the Digital Library Cover Profile Module Launch Page.
JAVASCRIPT	moduleScript.js	JavaScript file used to simulate the module launching on top of the Digital Library Cover Profile page.
OS X Disk Image File	Mongoose.dmg	Simple local web server that can be used on a Macintosh to run the module simulation locally.
Windows Executable	Mongoose.exe	Simple local web server that can be used on a Windows PC to run the module simulation locally.
Adobe Layered Art File	PlayScreen.psd	Adobe layered art file used as a template in the production of DLcover.jpg as described in Section 3.2.4 of this document.
Adobe Media	SBAC 1400x788.epr	Adobe Media Encoder Preset

Exemplar Instructional Module Technical Documentation

Encoder Preset		which can be used to down sample 1080p authored video to 1400x788 pixels for use in the Amplify Example Module UI.
Adobe Captivate Project File	SlideTest1.cptx	Adobe Captivate Project file which demonstrates how Captivate published content can be embedded and interact with the Example Module UI.
Windows Executable	TheoraConverter.NET 1.1 Setup.exe	Media encoder capable of creating OGG video files for Mozilla Firefox playback.
Adobe Layered Art File	thumbnail.psd	Adobe layered art file used as a template in the production of thumbnail.png as described in Section 3.2.3 of this document.
Word Document	Wrap-up Module Steps.docx	A short checklist of steps module developers should perform when finalizing a module which utilizes the Example Module UI.

TABLE 2: Contents of DLModulesFolder referenced in Table 1:

TYPE	NAME	DESCRIPTION
FOLDER	ExampleModule	Example Module Contents of this folder listed in TABLE 3 below.

TABLE 3: Contents of the ExampleModule in Table 2:

TYPE	NAME	DESCRIPTION
FOLDER	content	Folder which contains all published content for activities in the Example Module. Contents of this folder listed in TABLE 4 below.
FOLDER	dlcomponents	Folder containing files required by the Digital Library for Module Upload. Contents of this folder described in Section 3.2.4 of this document.
FOLDER	fonts	Folder containing all webfonts used by the Example Module.
FOLDER	images	Folder containing all images used to construct Example Module UI.
FOLDER	resources	Folder containing all supplementary resources that are referenced and downloadable through the Example Module.
FOLDER	scripts	Folder containing all JavaScript files necessary to run and configure a module. Contents of this folder described in TABLE 5 below.
FOLDER	styles	Folder containing all CSS Stylesheets used to construct module user interface.
HTML File	index.html	Main HTML file which runs example module.

TABLE 4: Contents of “content” folder from Table 3table:

TYPE	NAME	DESCRIPTION
FOLDER	FAProcess	Example video content delivered through UI.
FOLDER	ModuleOrientation	Second example of video content delivered through UI
FOLDER	SlideTest1	Example Captivate Published content

		delivered through Example Module UI.
--	--	--------------------------------------

TABLE 5: Content of “scripts” folder from Table 3:

TYPE	NAME	DESCRIPTION
JavaScript File	configuration.js	Module Configuration File described in Section 5.5 of this document.
JavaScript File	detect.js	Script file used to detect browser compatibility needs by a module.
JavaScript File	feedback.js	Script file containing functions used to perform module review in module production process.
JavaScript File	index.js	Script file containing all functions which run Example Module UI.
JavaScript File	jquery.min.js	JQuery v1.11.0 Library
JavaScript File	jquery-ui.min.js	Jquery UI v1.10.4 Library
JavaScript File	video.js	Script file used to provide playback controls in conjunction with video examples located in “content” folder previously described in this section.
JavaScript File	videoSub.js	Script file used in to provide subtitles in conjunction with video examples located in “content” folder previously described in this section.

5.2 Architectural Requirements

Beyond the standard module browser requirements listed above, the example module provided with this document makes use of <iframe> functionality which additionally requires:

- **All resources are served from the same domain** - The design of the Amplify example module utilizes communication between a parent user interface frame and a child content frame to deliver, update, and connect displayed content. Due to Cross-origin resource sharing - CORS (http://en.wikipedia.org/wiki/Cross-origin_resource_sharing) browser restrictions, the example module’s internal components must all be served from the same domain. The example modules will not run locally using file:// references in

Google Chrome without disabling local web security using the **--disable-web-security flag** when launching the browser.

This example module is designed to detect and notify a user attempting to launch the module from a local filesystem reference of this incompatibility.

5.3 Navigational Controls

While not a specific technical requirement when posting modules to the Digital Library, the example module provides a good illustration of typical navigational controls including:

- A Module SITEMAP - located on the top right of a module.
- LAUNCH-EXPLORE-SUMMARIZE navigation focused on the Module's connection to the Common Core State Standards and the Formative Assessment Process - located across the top of a module.
- Numbered activity buttons - located in the bottom right of a module.
- Next - Back Activity Navigation - located on the far left and right of a module.

Additionally, the example module provide playback and closed captioning controls located in the bottom left corner of a interface.



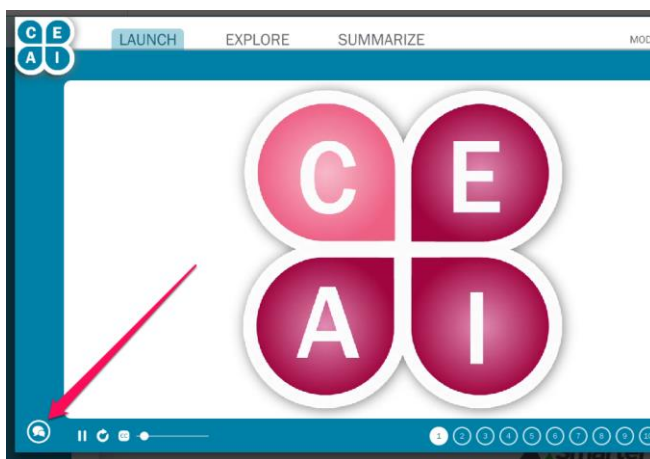
Example of a Module - Arrows point to navigational controls embedded in all Amplify authored modules.

5.4 Configuration Script

FILE LOCATION: SCRIPTS\Configuration.js

The configuration.js file is the primary file from which a module's internal navigation and linear (i.e. ordered numbered activities) learning experience is constructed. The configuration file begins with a few global variables which are described in code comments. Those configurable elements are:

feedbackMode (boolean) : enables/disables feedback mechanism in module (used in module production and Smarter Balanced Review).



moduleName (string) : String used when a user selects the PRINT option on an activity in a module.

E.G.

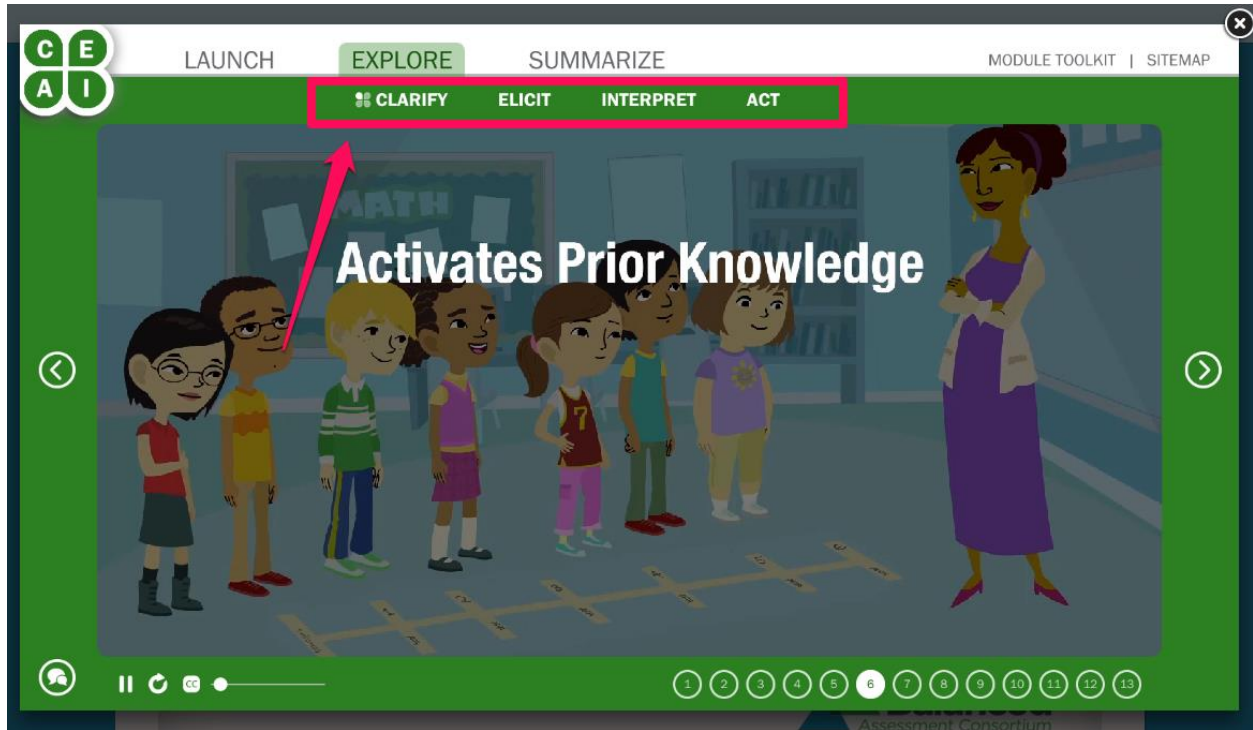
moduleName = "Understanding Fractions - Clarify Intended Learning"

feedbackModuleName (string) : String used to identify a module when submitting feedback in the module review process.

E.G.

feedbackModuleName = "MATH02"

showExploreSubMenu (boolean) : As Assessment Literacy modules do not use the four sections of the formative assessment process (i.e. the Explore sub-menu), disabling this value will hide it in the Explore section of a module.



The modPath Object

The modPath JSON object contains an array of elements which reference sub-folders of the **content** subfolder in the example module (described in Table 3 of Section 5.1 of this document).

Example of a modPath element:

```
{
  url: 'SlideTest1/', /* The folder in the content sub-folder that contains the content to frame (must contain index.html file) */
  delay:0, /* how long in ms to wait before loading this content */
  speed:400, /* how quickly to make the content move or reveal on screen */
  isCaptive: true,
  itemMap: [
```

```

    { title: 'Slide 2', color: '#2c8022', textColor: '#12530B', isSection: true, sectionName: 'Explore', indent: 15, runJS: function() {
    highlightFAPProcess(false, false, false, false, '#e38eb5', '#720e3d'); showIcon('Act'); }, playControls: false, pageIndex: 2 },
    { title: 'Slide 3', color: '#2c8022', textColor: '#12530B', isSection: false, sectionName: 'Explore', indent: 15, runJS: function() {
    highlightFAPProcess(false, false, false, false, '#e38eb5', '#720e3d'); showIcon('Clarify'); }, playControls: false, pageIndex: 3 }
    ]
  },

```

Each modPath element contains these variables:

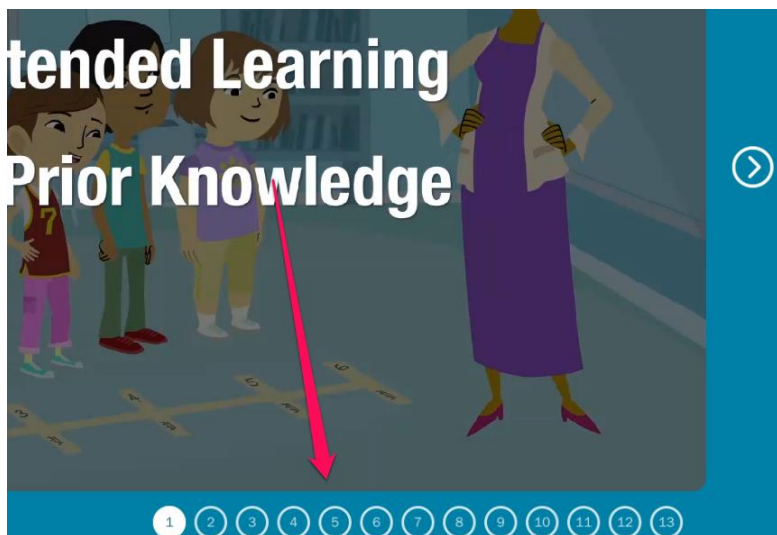
url (string include “/” after folder name): The represents the folder to reference for this element in the **content** folder of the module. It launches the index.html file in that folder.

delay (integer - milliseconds): This determines how long to delay loading that activity after the user clicks to load it.

speed (integer - milliseconds): This determines how quickly the activities cross fade.

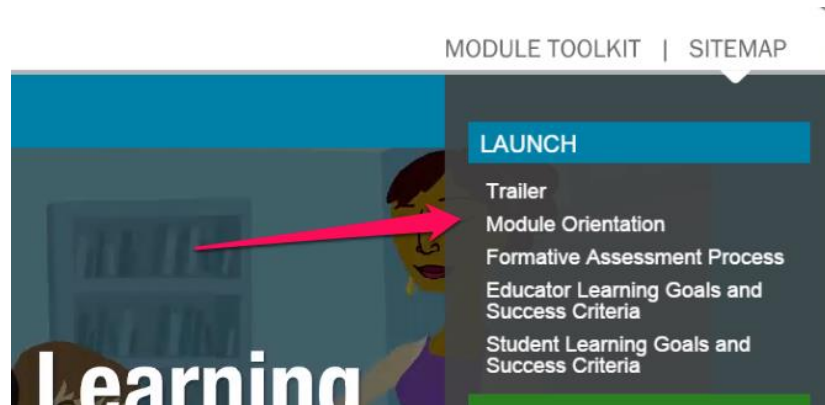
isCaptive (boolean): Captivate files have specific playback and content capabilities which are activated by setting this variable to “true”.

itemMap (array): This is an array of activities that are to be sequenced and provided activity numbers in the modules navigation:



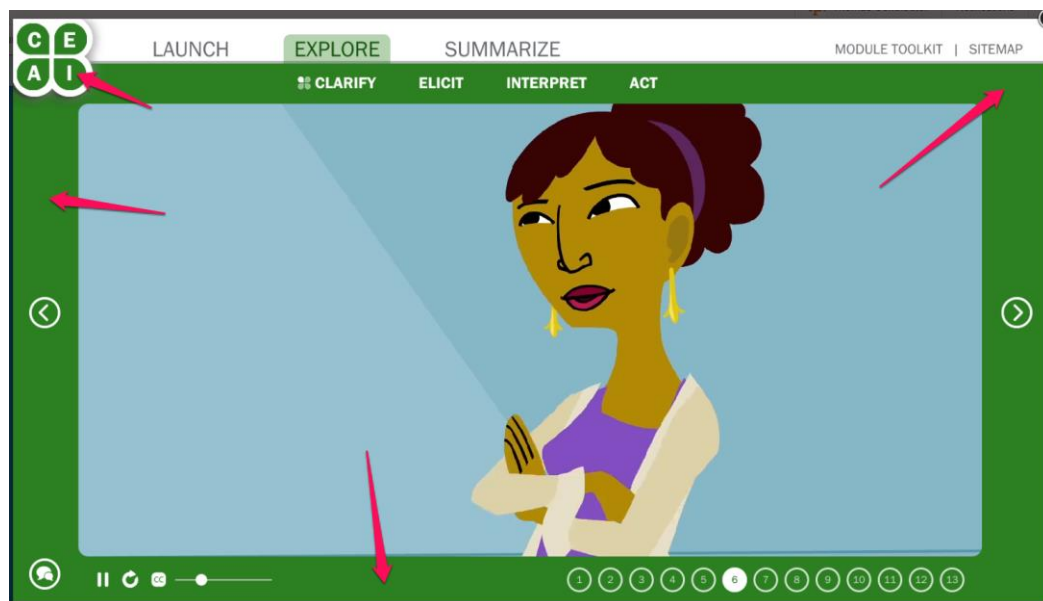
Each element in the **itemMap** array has it's own definition:

title (String): The title that appears on the SITEMAP



color (string - HTML Color Code with #): This is the color the interface should change to when the user is in that section. Standard colors are as follows:

- Launch: #0080a6
- Explore: #2c8022
- Summarize: #812F05
- Toolkit: #3b334d
- Assessment Literacy Modules: #a80f56

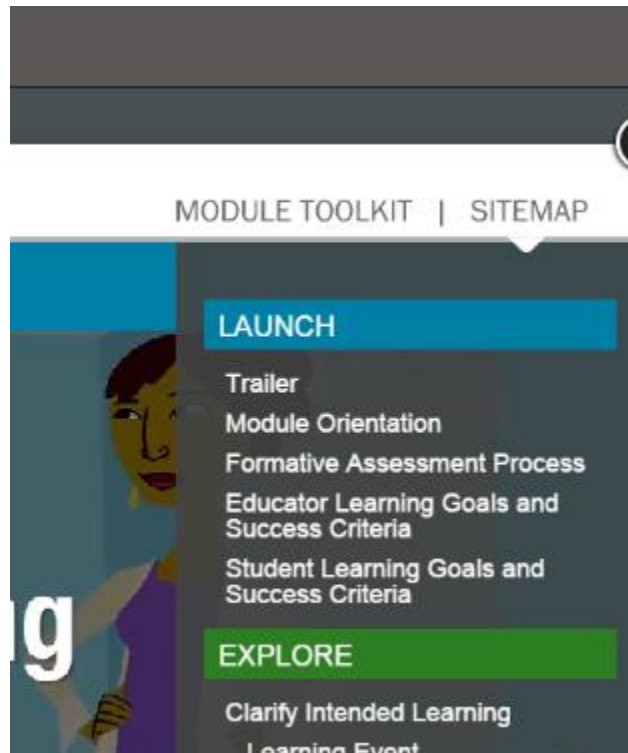


textColor (string - HTML Color Code with #): This is the color the system will use for certain navigational text when in this section. Standard colors are as follows:

- Launch: #1a6276
- Explore: #12530B
- Summarize: #812F05
- Toolkit: #3b334d
- Assessment Literacy Modules: #720e3d

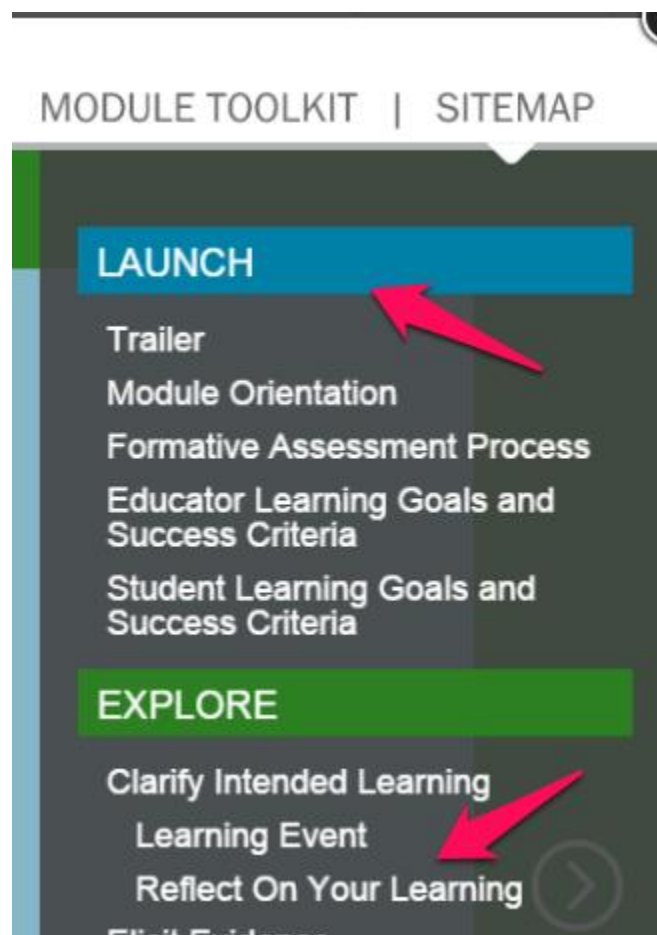


isSection (boolean) : When building the SiteMap and controlling the top navigation (i.e. Clicking Launch, Explore or Summarize) this allows the interface to identify the first element in a section. This should only be set to true for the FIRST element in the section, otherwise the SITEMAP will not be built correctly.



sectionName (string) : This is the section name used in conjunction with controlling top navigation and building the sections in the SITEMAP, valid values are (case sensitive): Launch, Explore, Summarize, Toolkit

indent (integer - pixels) : This represents how far to indent options on the SITEMAP. Default first level indent is 15 pixels, second is 30 pixels. An increment of 15 pixels will create the desired visual appearance.



runJS (javascript function) : This is a function that runs when this activity loads, there are a variety of options embedded in the main index.js script file that you can use for different purposes.

5.5 Available runJS JavaScript Functions

showIcon(faSectionName) - (Possible faSectionName Values : 'Clarify', 'Elicit', 'Interpret', 'Act') - This function moves the icon that sits in front of the explore sub-menu option the user is in.



highlightFAProcess(Clarify Boolean, Elicit Boolean, Interpret Boolean, Act Boolean, HighlightBackgroundColor, HighlightForegroundColor) - This function highlights the areas of the formative assessment process in the icon in the top left of the interface. Note if these areas are highlighted, the highlight must be inverted on every other page OR they will remain highlighted. Note this functionality is typically reserved for Assessment Literacy modules where a specific quadrant of the clover is highlighted.



playControls (boolean) : This variable tells the module if it should show or hide the play controls available in the module. Some module slides require playback because they are video or captivate slides that contain audio. Others are static and can have these controls hidden (e.g. the Toolkit).



pageIndex (integer) : This option pertains to captivate published files. If the activity that should be displayed is on a captivate slide in the referenced captivate publish, listing a slide number here and it will load the captivate file on that slide. If the activity it's not a captivate file, this value will be ignored and set to 1.

Captivate API Calls

While all JavaScript functions are available to Captivate, a few have been designed as API Calls available in Captivate to perform specific actions in the UI.

Available functions include:

callReviews() - No Inputs - Called to activate the review functionality of the module. Displays an alert box that instructs the user to close the module and access the review feature of the Digital Library.

callShare() - No Inputs - Similar to callReviews, called to activate the review functionality of the module. Displays an alert box that instructs the user to close the module and access the Share feature of the Digital Library.

callPrint(printTitle,printBody) - 2 inputs - Called to activate the print feature of a module. To activate, the function must be passed the desired page title that appears on the printed page and the page body. Supports HTML tags.

Example call: `parent.callPrint('Test Page Title','Hello World!');`

loadResource(resourceName) - 1 input - This function cross references the name sent with the second json object in the config file called resources. It will look for a match between the string you send it and the name field for each element, then grab the file from the resources folder of the module and present it in a new browser window.

saveValue(saveTo,value) - 2 inputs - Saves the value sent in the value field to the UI for later module access in the name submitted in the SaveTo field.

Example call: `parent.saveValue('Activity1',Text_Entry_Box_1_variable);`

loadValue(loadFrom, loadTo) - 2 inputs - Returns a value saved to the UI and places it in the submitted text input box. loadFrom is the value sent in the saveTo field when saving. loadTo in this case IS the Captivate text entry box's name.

Example call: `loadValue('Activity1',Text_Entry_Box_1);`

5.6 Module Review Feedback Mechanism

FILE LOCATION: SCRIPTS\feedback.js

The module production workflow included a Smarter Balanced Workgroup review for each module to assess both quality/accuracy of the presented educational material and technical compatibility/accessibility of each experience. To facilitate that experience a feedback submission interface was designed and embedded into the module user interface. Section 5.5 of this document provides instructions on how to activate/deactivate that interface.

This submission functionality utilizes asynchronous JavaScript calls which point to a REST endpoint located on an Amplify production server to receive reviewer feedback and log it to the Amplify production bug database. These asynchronous calls can be redirected to any other bug tracking platform that might be employed by other authoring teams.

Information available for submission to a feedback mechanism includes:

VideoLocation : Second counter when feedback button was pressed in video playback on a video activity in the module

feedbackName : Name of individual submitted with feedback

ScreenWH : Screen Width and Height

WindowWH : Browser Window Width and Height

BrowserShortString : A short string providing the browser name used by the end user

BrowserString : The browser's user agent string

feedbackModuleName : The name of the module used in bug submission

LocationTitle : The activity number of the location the submission was made at - mapped to circles at bottom of module

Critical : If the user checked the critical checkbox when submitting feedback

Functionality : If the user checked functionality checkbox when submitting feedback

Design : If the user checked design checkbox when submitting feedback

Content : If the user checked content checkbox when submitting feedback

Other : If the user checked other checkbox when submitting feedback

CommentField: The comment the user submitted when sending in feedback

OS : The OS the user was using when submitting feedback

5.7 Amplify Employed Content Production Techniques

Amplify module production employed a number of tools to expedite the authorship process while ensuring a high level of browser compatibility and accessibility. A number of those techniques are described below.

Adobe Captivate 7 Content Production

Adobe Captivate is an electronic learning tool for Microsoft Windows, and Mac OS X used to author interactive eLearning experiences while providing automated consideration for considerations including:

- Responsive design and HTML5 publishing
- A Visual Integrated Development Environment
- Cross Browser Compatability
- 508 and WCAG 2.0 Accessibility
- Rich multimedia integration

Interactive content in each module was constructed using Adobe Captivate. A number of similar eLearning Content Authoring development tools such as Articulate Storyline and Lectora Online provide an alternative approach to eLearning content authorship. One major consideration when selecting an authorship tool is compliance with the W3C Authoring Tool Accessibility Guidelines (ATAG) 2.0 available here: <http://www.w3.org/TR/ATAG20/>.

Mongoose Simple Web Server

Included in the example module is a small application called Mongoose which will allow a designer to access a module locally by providing HTTP access to a folder from a local web server running on port 8080. Doing so allows the designer to direct their browser to <http://localhost:8080/> to simulate a module.

Video Production



Smarter Balanced Digital Library

Exemplar Instructional Module Technical Documentation

Video was authored at 1080p and then reduced via an Adobe Encoder Preset to a 1400 by 788 pixel mp4 video file. Video files were subsequently encoded to OGG using Theora Converter to support Mozilla Firefox video playback.

A current version of the open source Theora product as well as the Adobe Media Encoder preset for resizing and compressing the video to MP4 are included in the example module and described in Section 5.1 of this document.

File names:

SBAC 1400x788.epr
TheoraConverter.NET 1.1 Setup.exe

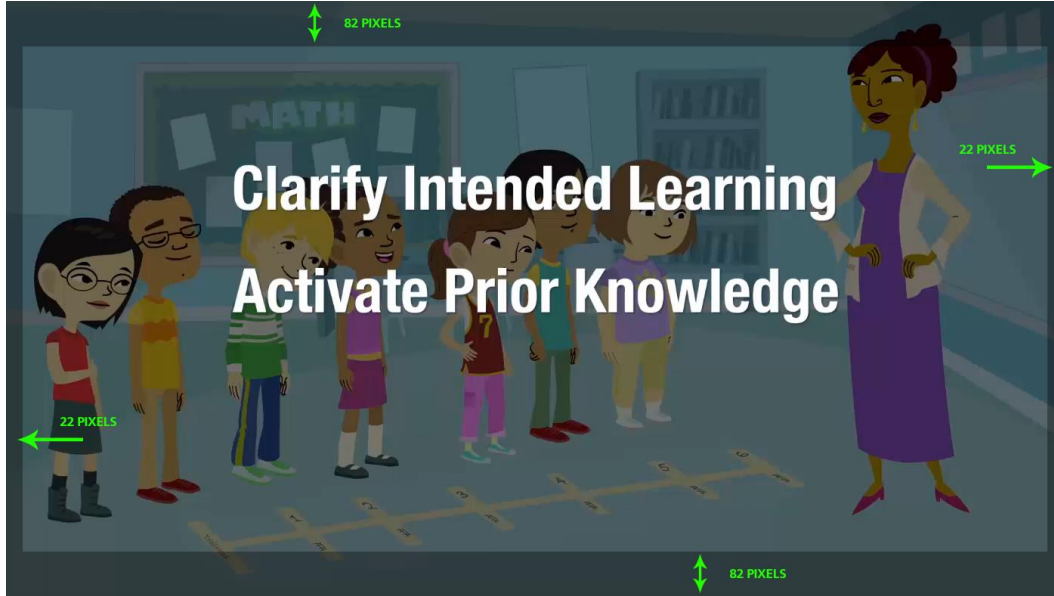
That preset is provided in the example module and described in Section 5.1 of this document.

Video Production Safe Areas

The example module UI requires a safe area of 22 pixels on the left and right of the video and 82 pixels on the top and bottom while authoring at 1920px by 1080px (i.e. 1080p).

Ensuring essential visuals or text on screen does not occur inside the safe area pixel counts listed above will guarantee all users will have access to the content.

Description of Video Safe Area: [http://en.wikipedia.org/wiki/Safe_area_\(television\)](http://en.wikipedia.org/wiki/Safe_area_(television))

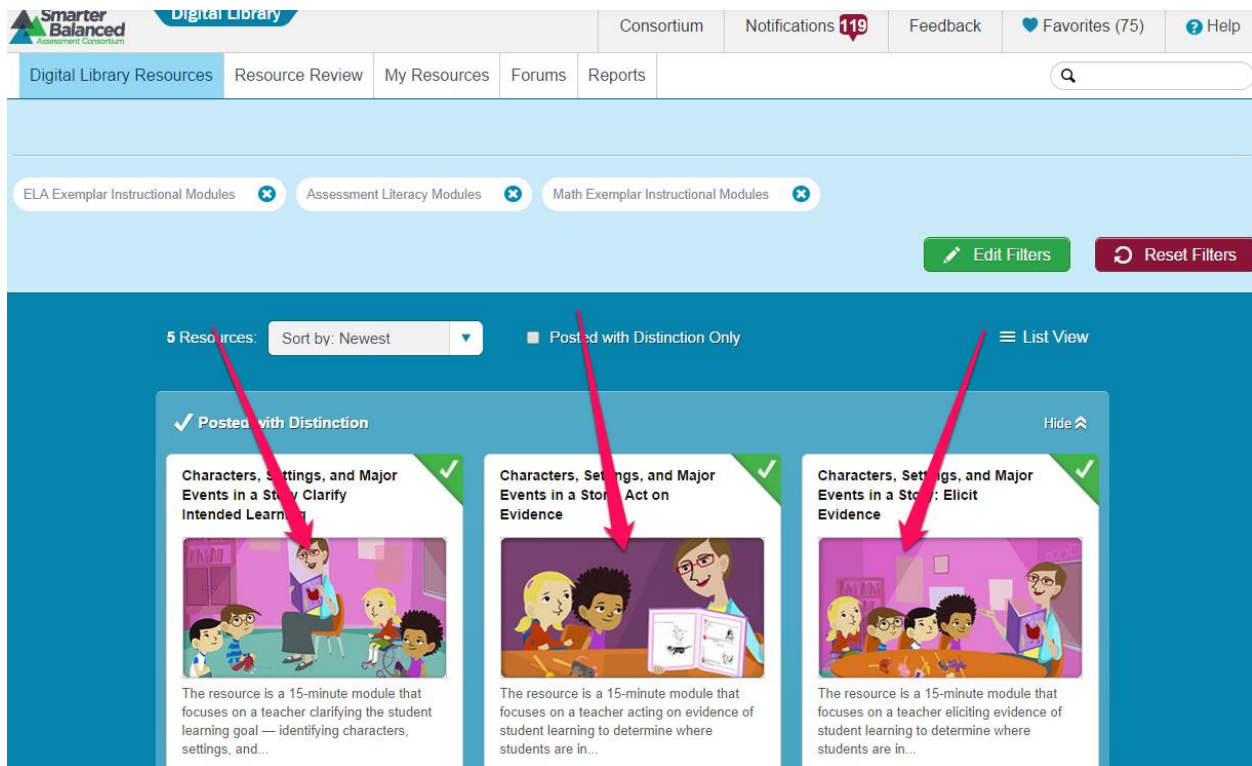


Playscreen Production



PlayScreen.psd is described in Section 5.1 and provides a layered art file that can be used as a template when building the DLCover.jpg file described in Section 3.2.4.

Thumbnail Production



PlayScreen.psd (Section 5.1 Table 1) provides a layered art template that can be used when building the DLcover.jpg file described in Section 3.2.4 of this document.

Thumbnail.psd (Section 5.1 Table 1) provides a layered art template that can be used when building the thumbnail.png file described in Section 3.2.4 of this document.

Video Captioning

FILE LOCATION: SCRIPTS\videoSub.js

Captions.html is an SRT caption file generated using Techsmith Camtasia (<http://www.techsmith.com/camtasia.html>). SRT caption files are exported and renamed to .html extension. This JavaScript Polyfil for captioning was utilized to ensure browser compatibility with older browsers which do not natively support the WebVTT/SRT captioning format.



Smarter Balanced Digital Library
Exemplar Instructional Module Technical Documentation
