

EXTENDING ADOBE CAPTIVATE WITH JAVASCRIPT

A WEB DEVELOPER'S PERSPECTIVE

AUDIENCE

- Learning interaction designers
- Project managers / Course strategy developers
- Web Developers
- eLearning methodology strategists
- Content Authors

CONTEXT

- Captivate
- HTML projects
- “Responsive” design
- Windows 10 development environment
- JavaScript ECMA 2015
- Chrome browser
- Notepad++ text editor

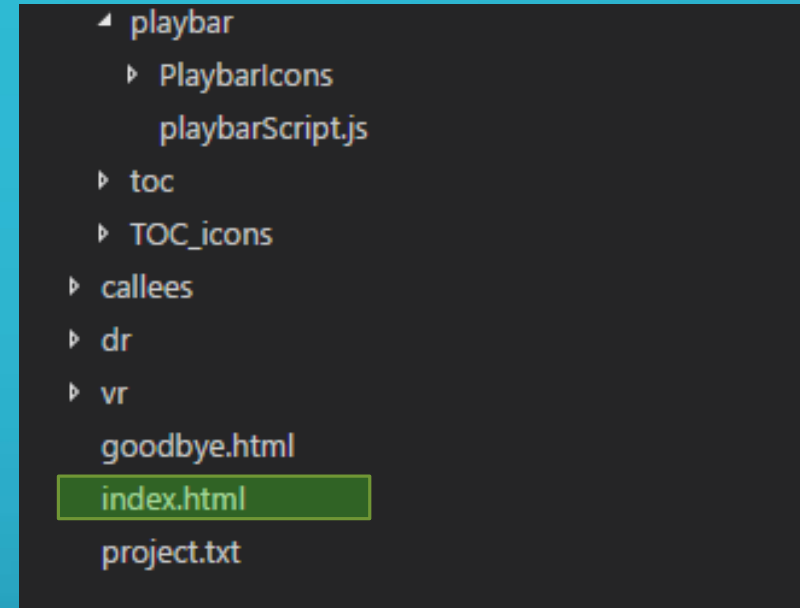
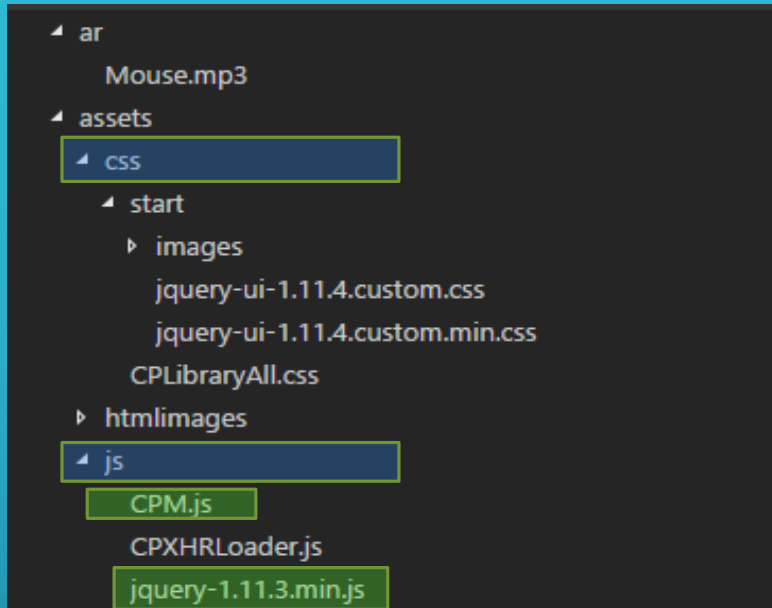
PLAN

- Captivate as a web development platform
- JavaScript as a development tool
- Efficient development of JavaScript/Captivate scripts
 - External JavaScript files
 - Debugging in Chrome
- Example Script
 - Fully custom quiz interactions
 - Full-screen mode
- Overview of other possibilities with JavaScript
- Adobe documented vs. undocumented functions
 - Decompressing CPM.js
- The CPM.js file and implications for development
 - Bridging between JavaScript and Captivate
- Questions

CAPTIVATE FROM THE WEB DEVELOPERS PERSPECTIVE

- WYSIWYG website builders:
 - “Closed” builders generate sites that cannot easily be modified after being generated
 - Easy to get started building, limited access to potential of modern design
 - Weebly, Wix, Squarespace
 - “Open” builders support direct modification of generated sites & continued editing
 - Deeper understanding of web technologies needed
 - Pinegrow, Bootstrap Studio, Bootply
- Captivate – 90% closed / 10% open
- Custom features valuable for eLearning
- Reasonable strategy given initial target audience

ANATOMY OF A WEBSITE (CAPTIVATE FILE LAYOUT)



- A module produced by Captivate is structured in a very common website design style
- A zipped module is simply a single-file version of this exact directory structure
- When a captivate module is loaded into an LMS, the zip file is simply uncompressed by the LMS
- Websites typically need to be “served” by a server program (apache/nginx) in case external content needs to be loaded
- When all content is inside the module directory, a browser can be used to view the website

ANATOMY OF A CAPTIVATE WEBSITE

HTML

```
<body>
  <div id="mobile-sidebar" class="visible-xs 1
    <a class="mobile-sidebar-header" href="#"
      {{template "cogname" .}}
    </a>
    <div class="mobile-sidebar-content">
      <ul class="frow column-start">
        <ul class="frow centered menuSel
          <li><a href="#overview">Over
          <li><a href="#service">Servi
          <li><a href="#customers">Cus
          <li><a href="#about">About</
        </ul>
      </ul>
    </div>
  </div>
  <div id="click-cover" class="visible-xs lets
```

CSS

```
{
  background:url('../Playbar_icons/Play_icon.png
  width:58px;
  height:59px;
  float:left;
  position:absolute;
  left:50px;
}
.playButton: hover
{
  background:url('../Playbar_icons/Play_icon.png
  width:58px;
  height:59px;
  float:left;
  position:absolute;
  left:50px;
}
```

Javascript

```
};
(function (i, m) {
  var b = function (a, c) {
    return new b.Instance(a, c || {})
  };
  b.defaults = {
    stop_browser_behavior: {
      userSelect: "none",
      touchAction: "none",
      touchCallout: "none",
      contentZooming: "none",
      userDrag: "none",
      tapHighlightColor: "rgba(0,0,0,0)"
    }
  };
  b.HAS_POINTEREVENTS = i.navigator.pointerE
  b.HAS_TOUCHEVENTS = "ontouchstart" in i;
  b.MOBILE_PREFIX = (mobile|tablet|ip(ad|hone
```

- Same structures are seen in Captivate as in all websites
- “CPM.js” file contains
 - All content data – shapes, text, timing, placement, quiz
 - Captivate JavaScript Library that “runs” the website
 - Since the file is compressed, it is hard to decipher

WHY JAVASCRIPT?

- Most popular programming language – StackOverflow / Github
- Used for both user interaction in browser and business logic on server
 - Access all the power of the browser
- Completely free development environment
- All Browsers have powerful, built-in debugging tools
- Very fast design/test cycle - no “publishing/compiling” process
- Most profound change in learning process – learning on demand
 - Stackoverflow <http://stackoverflow.com/insights/survey/2016>
 - 2.7Million questions, 3.2Million answers in 2015
 - Thousands of tutorials

WHY USE JAVASCRIPT WITH CAPTIVATE

Upside

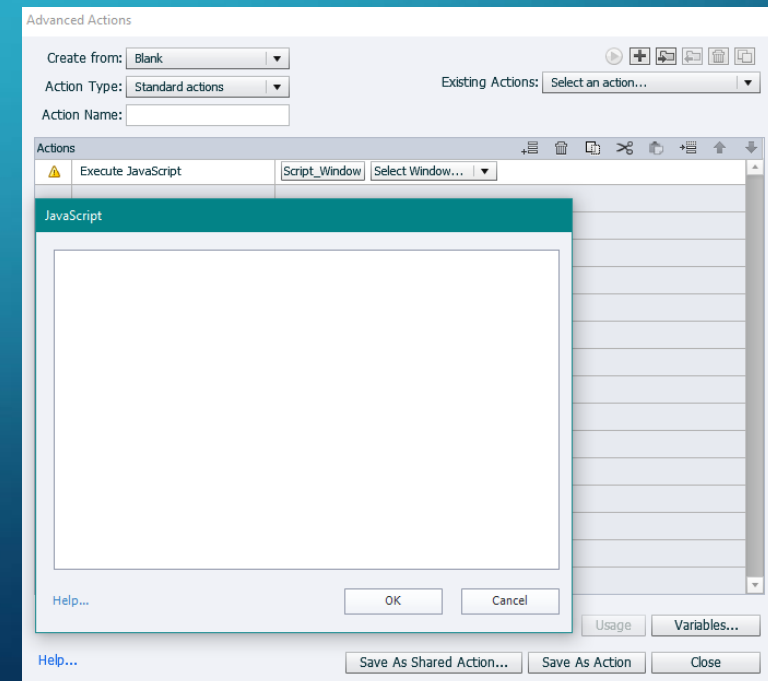
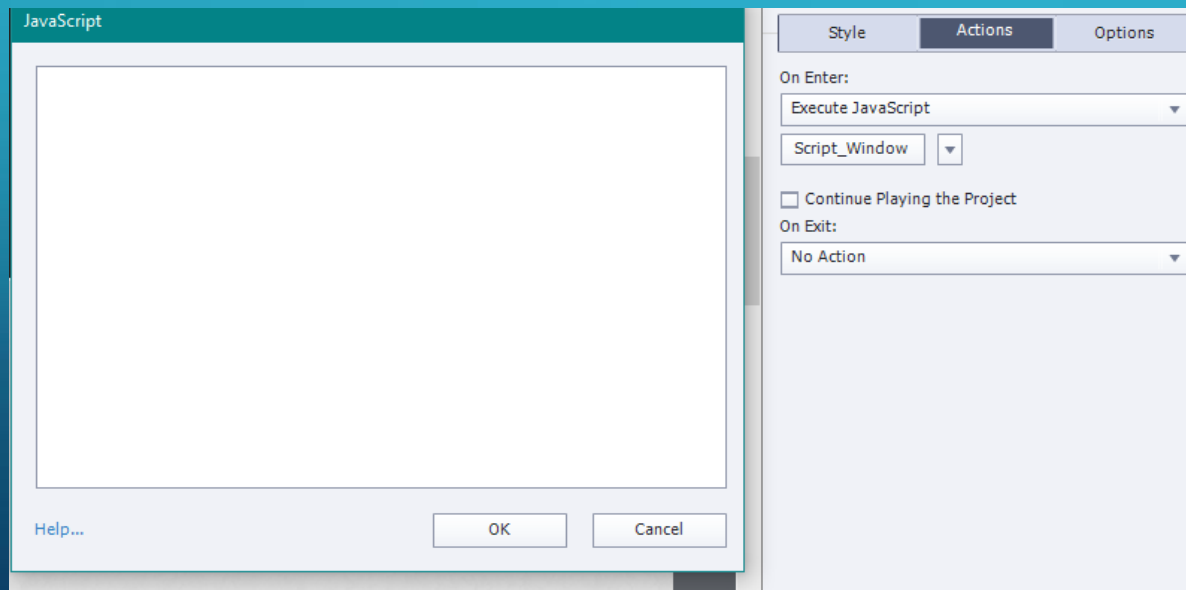
- You can control the entire user interface with JavaScript
 - Change shape properties, display notifications, react to any user event
 - Create custom quiz interactions, unique animations etc..
- JavaScript functions can be debugged while the presentation is running, unlike advanced actions
- With a trick, JavaScript functions can be updated/modified without “re-publishing”
...fast development turnaround
- Many online tutorials for using JavaScript with Captivate
 - Large subject area, no tutorial is can be comprehensive – point solutions and examples

Downside

- Steeper learning curve
- Lots of cool stuff is undocumented by Adobe, discovered and published by developers

HOW TO WORK EFFICIENTLY WITH JAVASCRIPT

- Internally supported approach: Use built-in JavaScript script window
 - No syntax checking
 - Must re-publish module to update
 - Hard to maintain, code is sprinkled throughout the modules



HOW TO WORK EFFICIENTLY WITH JAVASCRIPT

- External file holds JavaScript functions
- JavaScript file is located outside the module so it won't be deleted when the module is updated
- JavaScript file is typically in the same directory that holds the module
- **To connect external javascript file, add to the first slide “On enter execute JavaScript”**

```
$('#body').append('<script src="../multichoice.js" async="false"></script>');
```

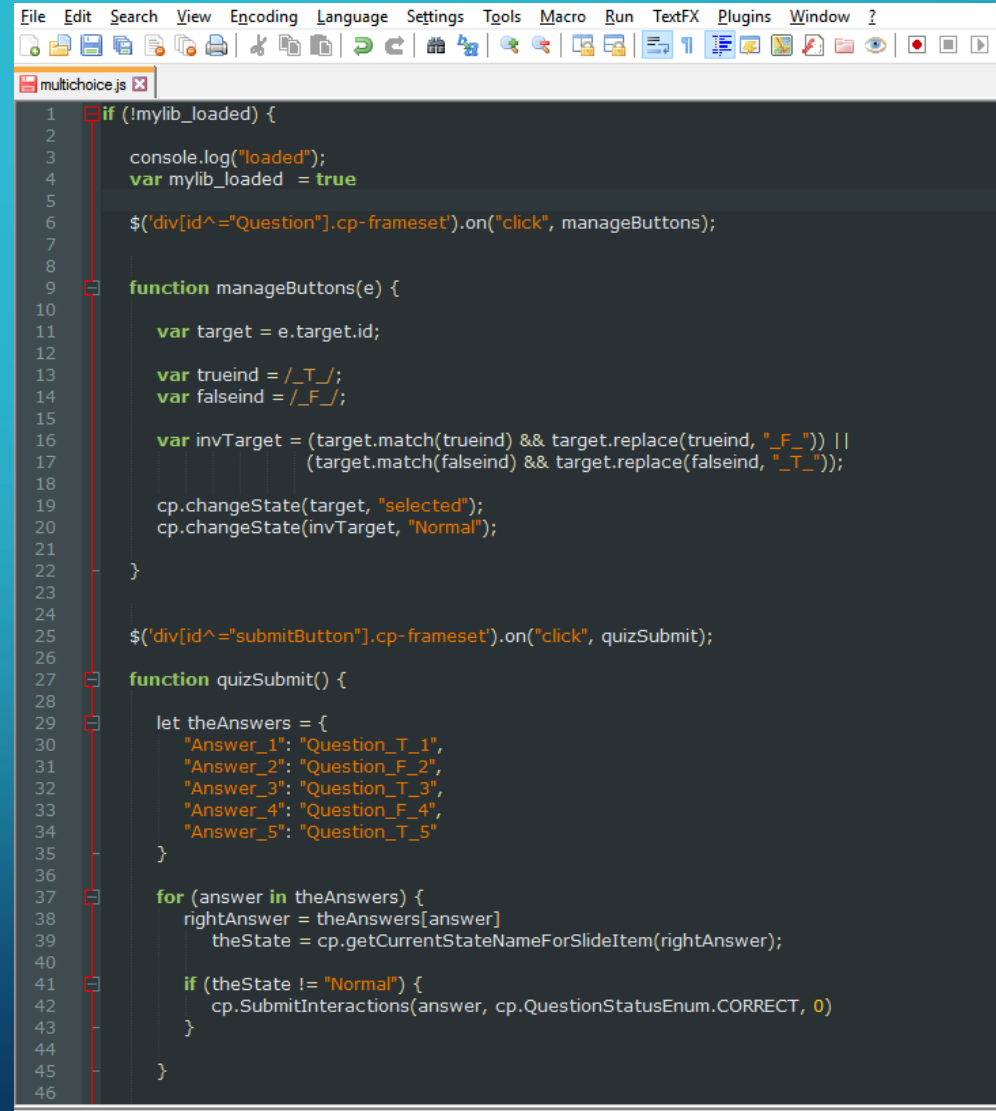
- Changes in this file will be loaded whenever the module is viewed, no need to re-publish course.
- Downside – files “outside” a module are only accessible when using http:// not file://
 - Use local web server
 - Move file inside module

HOW TO WORK EFFICIENTLY WITH JAVASCRIPT

Notepad++ text editor as example

Far easier than built-in script window!

- JavaScript syntax and error highlighting
- Multiple windows, spell check etc.

A screenshot of the Notepad++ text editor interface. The title bar shows 'multichoice.js'. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, TextFX, Plugins, Window, and ?. The toolbar contains various icons for file operations and editing. The code is written in JavaScript and features syntax highlighting: keywords like 'if', 'function', 'var', 'let', 'for', and 'if' are in blue; strings are in red; comments are in green; and identifiers are in black. The code is as follows:

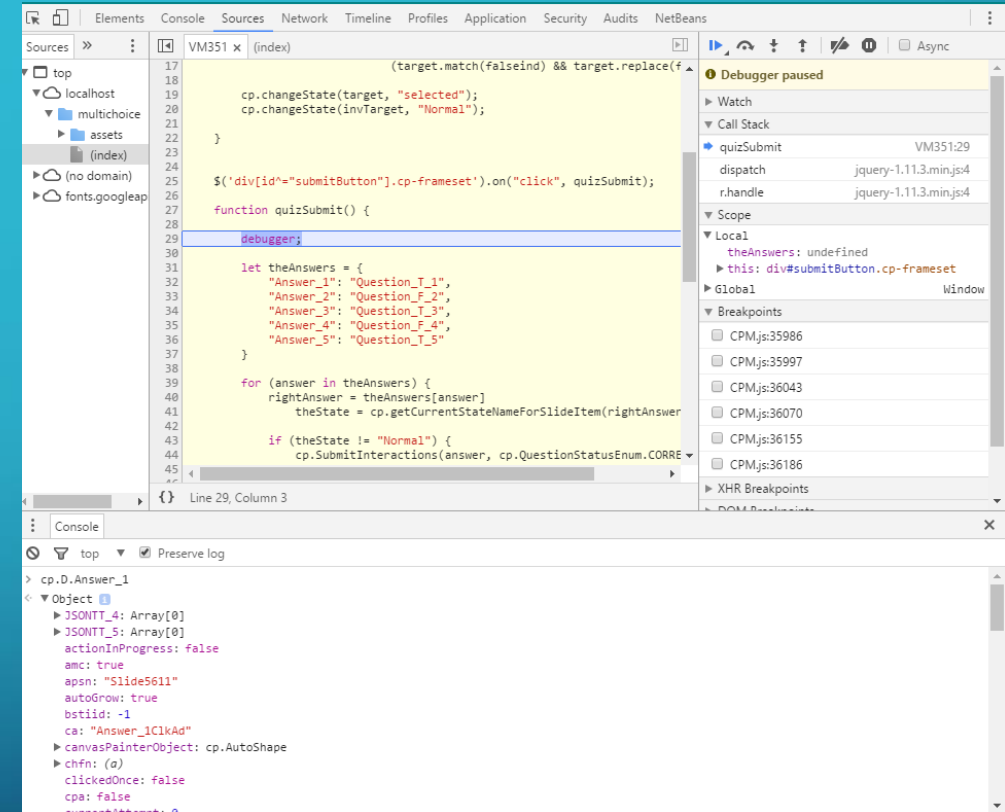
```
1  if (!mylib_loaded) {  
2  
3      console.log("loaded");  
4      var mylib_loaded = true  
5  
6      $('div[id^="Question"]').cp-frameset().on("click", manageButtons);  
7  
8  
9  }  
10  
11  function manageButtons(e) {  
12  
13      var target = e.target.id;  
14      var trueind = /_T_/;  
15      var falseind = /_F_/;  
16  
17      var invTarget = (target.match(trueind) && target.replace(trueind, "_F_")) ||  
18                      (target.match(falseind) && target.replace(falseind, "_T_"));  
19      cp.changeState(target, "selected");  
20      cp.changeState(invTarget, "Normal");  
21  
22  }  
23  
24  
25  $('div[id^="submitButton"]').cp-frameset().on("click", quizSubmit);  
26  
27  function quizSubmit() {  
28  
29      let theAnswers = {  
30          "Answer_1": "Question_T_1",  
31          "Answer_2": "Question_F_2",  
32          "Answer_3": "Question_T_3",  
33          "Answer_4": "Question_F_4",  
34          "Answer_5": "Question_T_5"  
35      }  
36  
37      for (answer in theAnswers) {  
38          rightAnswer = theAnswers[answer]  
39          theState = cp.getCurrentStateNameForSlideItem(rightAnswer);  
40  
41          if (theState != "Normal") {  
42              cp.SubmitInteractions(answer, cp.QuestionStatusEnum.CORRECT, 0)  
43          }  
44  
45      }  
46  }
```

DEBUGGING JAVASCRIPT WITH CHROME

F12 opens Chrome debugger!

```
function quizSubmit() {  
  debugger;  
  
  let theAnswers = {  
    "Answer_1": "Question_T_1",  
    "Answer_2": "Question_F_2",  
    "Answer_3": "Question_T_3",  
    "Answer_4": "Question_F_4",  
    "Answer_5": "Question_T_5"  
  }  
  
  for (answer in theAnswers) {  
    rightAnswer = theAnswers[answer]  
    theState = cp.getCurrentStateNameForSlideItem(rightAnswer);  
  
    if (theState !== "Normal") {  
      cp.SubmitInteractions(answer, cp.QuestionStatusEnum.CORRECT, 0)  
    }  
  }  
  
  cpCmdNextSlide = 1;  
}
```

← Pauses execution of function, enables complete debugging environment in Chrome



Step-by-step debugging – unlike advanced actions

EXAMPLE – CUSTOM QUIZ INTERACTION

[HTTPS://GITHUB.COM/SDWARWICK/CAPTIVATE-DEMOS](https://github.com/sdwarwick/captivate-demos)

Rules:

No scoring until “Submit” is pressed

True/false toggles correctly

Score for each answer may be different

+25 points for 4/5 right answers

+50 points for 5/5 right answers

Strategy:

All of the user interactions managed by
JavaScript

Quiz will be scored and submitted by JavaScript

	T	F
Mares eat oats	<input type="radio"/>	<input type="radio"/>
Cow's have hiccups	<input type="radio"/>	<input type="radio"/>
Does eat oats	<input type="radio"/>	<input type="radio"/>
Birds snore	<input type="radio"/>	<input type="radio"/>
Little lambs eat ivy	<input type="radio"/>	<input type="radio"/>

Submit

EXAMPLE – CUSTOM QUIZ INTERACTION

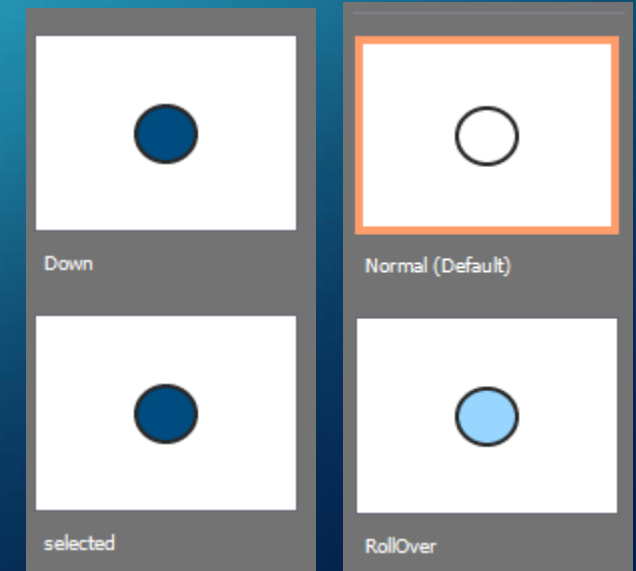
- Slide “on enter execute JavaScript”:

```
$('#body').append('<script src="../multichoice.js" async="false"></script>');
```

```
var fontLink = '<link href="https://fonts.googleapis.com/css?family=Calligraffiti" rel="stylesheet">';  
$(fontLink).appendTo("head");
```

- All buttons are simple circle smartshapes with “use as button”

- Add an additional state called “selected”
- This will be controlled by JavaScript



EXAMPLE – CUSTOM QUIZ INTERACTION

- The shapes are labeled using a regular pattern that will be easily distinguished in the JavaScript Code
- The hidden answer buttons are all set to “Include in Quiz” and points can be assigned to each answer
- Add variables to enable connection between JavaScript and Captivate
- That’s it.. no advanced actions

The screenshot displays a quiz interface with two columns of radio button questions labeled 'T' (True) and 'F' (False). Below the questions is a green 'Submit' button labeled 'submitButton'. To the right, a table lists 'Hidden answer Button ID' and 'Hidden answer Button Quiz Point Value'.

Hidden answer Button ID	Hidden answer Button Quiz Point Value
Answer_1	10
Answer_2	10
Answer_3	10
Answer_4	10
Answer_5	10
Bonus_25	25
Bonus_50	25

```
baseMaxScore
baseScore
bonusMaxScore
bonusScore
cpQuizInfoStudentID
cpQuizInfoStudentName
numberOfQuestions
numberOfRightAnswers
```


EXAMPLE CUSTOM QUIZ INTERACTION - TOGGLE

```
$('#div[id^="Question"].cp-frameset').on("click", manageToggleButtons);

function manageToggleButtons(clickedButtonObject) {

    // get shpe id of the clicked button, this will be "selected"
    var targetID = clickedButtonObject.target.id;

    // create the name of the button you need to toggle to "unselected"
    if ( targetID.match(/_T_/) ) {
        var invTargetID = targetID.replace(/_T_/, "_F_")
    }

    if ( targetID.match(/_F_/) ) {
        invTargetID = targetID.replace(/_F_/, "_T_")
    }

    // captivate undocumented function to change state of object
    cp.changeState(targetID, "selected");
    cp.changeState(invTargetID, "Normal");
}
```

Single line in javascript says:

Find all buttons that start with the word “Question”. When clicked, call “manageToggleButtons function

Take the name of the button that was pressed, changes any “_T_” to “_F_” and any “_F_” to “_T_”

It then calls an undocumented captivate function “cp.changeState” to toggle between the “Normal” view and the “selected” view

Over the years, many people have contributed to weeding through the CPM.js code to find these functions

EXAMPLE CUSTOM QUIZ INTERACTION - SCORING

```
$('#div[id^="submitButton"].cp-frameset').on("click", quizSubmit);

function quizSubmit() {

    //debugger;

    // these are defined in captivate and used in analysis
    numberOfRightAnswers = 0;
    numberOfQuestions = 0;
    baseScore = 0;
    baseMaxScore = 0;
    bonusScore = 0;
    bonusMaxScore = 0;

    // the right answer button is selected, signal this internal button
    var theRightAnswers = {
        "Question_T_1" : "Answer_1",
        "Question_F_2" : "Answer_2",
        "Question_T_3" : "Answer_3",
        "Question_F_4" : "Answer_4",
        "Question_T_5" : "Answer_5"
    }
}
```

- The first line triggers the quiz submit function for the button with the ID “submitButton”
- Variables defined in captivate can be directly used in JavaScript!
- The correct answers are defined by which of the question buttons were set to state “selected”
- If the correct answer is selected, which hidden button should be activated?

EXAMPLE CUSTOM QUIZ INTERACTION - SCORING

```
// the right answer button is selected, signal this internal button
var theRightAnswers = {
  "Question_T_1" : "Answer_1",
  "Question_F_2" : "Answer_2",
  "Question_T_3" : "Answer_3",
  "Question_F_4" : "Answer_4",
  "Question_T_5" : "Answer_5"
}

//check each of the right answer button for state, if selected, signal to captivate
for (rightAnswerButton in theRightAnswers) {

  numberOfQuestions = numberOfQuestions + 1;
  rightAnswerSenderButton = theRightAnswers[rightAnswerButton];

  // get quiz value for this answer - this is obscure but works
  answerObjectID = cp.D[rightAnswerSenderButton].qnq;
  answerValue = cp.D[rightAnswerSenderButton + "q" + answerObjectID].w;

  //add to max base score
  baseMaxScore = baseMaxScore + answerValue;

  theState = cp.getCurrentStateNameForSlideItem(rightAnswerButton);

  if (theState == "selected") {
    // undocumented function for signalling to a quiz button
    cp.SubmitInteractions(rightAnswerSenderButton, cp.QuestionStatusEnum.CORRECT, 0)
    numberOfRightAnswers = numberOfRightAnswers + 1;
    baseScore = baseScore + answerValue;
  }
}
```

When writing code, try to keep things flexible..

- Determine maximum number of questions, maximum score, answered questions and score values on the fly
- Here's how to get the value of a quiz button
- Here's how to find the state of a slide object
- If the right button was selected then we call another undocumented function that signals to captivate that an answer was given correctly.

EXAMPLE CUSTOM QUIZ INTERACTION - SCORING

```
// add bonuses
rightAnswerSenderButton = "Bonus_25"
answerObjectID = cp.D[rightAnswerSenderButton].qnq;
answerValue = cp.D[rightAnswerSenderButton + "q" + answerObjectID].w;
bonusMaxScore = bonusMaxScore + answerValue;

if (numberOfRightAnswers >= 4) {
    cp.SubmitInteractions(rightAnswerSenderButton,
        cp.QuestionStatusEnum.CORRECT, 0);
    bonusScore = bonusScore + answerValue;
}

rightAnswerSenderButton = "Bonus_50"
answerObjectID = cp.D[rightAnswerSenderButton].qnq;
answerValue = cp.D[rightAnswerSenderButton + "q" + answerObjectID].w;
bonusMaxScore = bonusMaxScore + answerValue;

if (numberOfRightAnswers == 5) {
    cp.SubmitInteractions(rightAnswerSenderButton,
        cp.QuestionStatusEnum.CORRECT, 0);
    bonusScore = bonusScore + answerValue;
}

cpCmndNextSlide = 1;
```

Find quiz value for the bonus points by looking at the Captivate data

Award points based on some criteria - here it is at least 4 answers right

Here it is 5 answers right...

After we are done, we signal to move to next slide by simply setting the "next slide" flag variable

EXAMPLE – CUSTOM QUIZ INTERACTION

Why is this example important?

- Other than labeling the buttons, setting quiz values and loading the external JavaScript module, no advanced actions or special processing is needed
- The scoring is completely general. Any set of button presses can be used to generate a specific quiz result
- Scoring doesn't happen for any of the quizzing until the interaction is complete
- Custom interactions need not be limited to one “slide”
- Although not shown, at any point in the process, additional information can be given to the user
- Other measures can be made along the way:
 - How many times has the user changed their score?
 - How long did it take before the user completed the quiz?

EXAMPLE – “FULL SCREEN” MODE

- Any button that has a name starting in “fullscreen” will activate this code
- Also works for presentations embedded in other applications (IFRAME)

```
function fullScreenButton() {  
  let j = $('[id^="fullscreen"]').on('click', function (e) {  
    let i = parent.document.getElementsByTagName("iframe")[0]  
    if (i == null) {  
      i = document.getElementById("main_container")  
    }  
    i.requestFullscreen && i.requestFullscreen();  
    i.webkitRequestFullscreen && i.webkitRequestFullscreen();  
    i.mozRequestFullscreen && i.mozRequestFullscreen();  
    i.msRequestFullscreen && i.msRequestFullscreen();  
  });  
};
```

```
function cancelFullScreenButton() {  
  let j = $('[id^="stdscreen"]').on('click', function (e) {  
    let i = parent.document;  
    if (i == null) {  
      i = document.getElementById("main_container")  
    }  
    i.cancelFullScreen && i.cancelFullScreen();  
    i.webkitCancelFullScreen && i.webkitCancelFullScreen();  
    i.mozCancelFullScreen && i.mozCancelFullScreen();  
    i.exitFullscreen && i.exitFullscreen();  
  });  
};
```

```
fullScreenButton();  
cancelFullScreenButton();
```

WHAT ELSE DOES JAVASCRIPT OPEN UP?

- References to external content – fonts, libraries
- Real-time, group interactions with backend data sources (AJAX)
- Video game-level animations
- Dynamic Graphing and Charting
- Fine-grained experience measurement
- Pass information between parent/child windows
- Custom reporting to LMS/LRS
- Access to the entire web development community!

UNDOCUMENTED CAPTIVATE FUNCTIONS AND DATA STRUCTURES..

The CPM.js library

- 25,000 JavaScript statements in the basic library to “run” a presentation
- 100,000+ statements to define all objects in a large presentation

CPM.js defines 100+ “top level objects/properties”

CP object - defines 751 objects/properties

CP.D - all of the slide objects and quizzing information

CP.DD - drag/drop interaction data

Lots of other things, too much to even begin to describe..

- Animation
- Display timing
- Quiz handling
- Drag/Drop interactions
- LMS Reporting system..

CPM.js code is well organized with very descriptive top level function names

105 “TOP LEVEL” VARIABLES GENERATED BY CPM.JS

cp

cpXHRJSLoader

cpAPIInterface

cpAPIEventEmitter

cpCmdndVolume

cpCmdndMute

cpCmdndCC

cpCmdndNext

cpCmdndNextSlide

cpCmdndPrevious

cpCmdndNextOnReview

cpCmdndPreviousSlide

cpCmdndPreviousOnReview

cpCmdndPlaybarMoved

cpCmdndShowPlaybar

cpCmdndFastForward

cpCmdndRewindAndPlay

cpCmdndRewindAndStop

cpCmdndGotoFrame

cpCmdndGotoFrameAndResume

cpCmdndGotoSlide

cpCmdndGotoSlideAndResume

cpCmdndGotoSlideByUIDAndResume

cpCmdndResume

cpCmdndPause

cpCmdndExit

cpLockTOC

cpCmdndInfo

cpCmdndTOCVisible

cpInfoSlidesInProject

cpInfoFPS

cpInfoAuthor

cpInfoCompany

cpInfoEmail

cpInfoWebsite

cpInfoCopyright

cpInfoProjectName

cpInfoDescription

cpInfoCurrentFrame

_cpInfoCurrentFrame

cpInfoPrevFrame

cpInfoFrameCount

cpInfoPrevSlide

_cpInfoPrevSlide

cpInfoLastVisitedSlide

_cpInfoLastVisitedSlide

cpInfoCurrentSlide

cpInfoCurrentSlideIndex

_cpInfoCurrentSlide

cpInfoCurrentSlideLabel

_cpInfoCurrentSlideLabel

cpInfoSlideCount

cpInfoStandalone

cpInfoHasPlaybar

cpInfoCurrentSlideType

cpInfoResultSlide

cpInfoElapsedTimeMS

cpInfoEpochMS

cpInfoCurrentMinutes

cpInfoCurrentHour

cpInfoCurrentTime

cpInfoCurrentDay

cpInfoCurrentYear

cpInfoCurrentMonth

cpInfoCurrentDate

cpInfoCurrentDateString

cpInfoCurrentDateStringDDMMYYYY

cpInfoCurrentLocaleDateString

cpCmdndGotoQuizScopeSlide

cpQuizInfoLastSlidePointScored

cpQuizInfoQuestionSlideType

cpQuizInfoAnswerChoice

cpQuizInfoMaxAttemptsOnCurrentQuestion

cpQuizInfoPointsPerQuestionSlide

cpQuizInfoNegativePointsOnCurrent
QuestionSlide

cpQuizInfoQuestionSlideTiming

cpQuizInfoQuizPassPoints

cpQuizInfoQuizPassPercent

cpQuizInfoTotalProjectPoints

cpQuizInfoTotalUnansweredQuestions

cpQuizInfoNoQuestionsPerQuiz

cpQuizInfoPointsscored

cpQuizInfoPretestPointsscored

cpQuizInfoPretestScorePercentage

cpQuizInfoTotalCorrectAnswers

cpInfoPercentage

cpQuizInfoTotalQuizPoints

cpQuizInfoAttempts

cpQuizInfoTotalQuestionsPerProject

cpQuizInfoQuestionPartialScoreOn

cpQuizScopeSlide

cpInfoQuizScope

cpQuizInfoPassFail

cpInfoCourseID

cpInfoCourseName

cpQuizInfoPreTestTotalCorrectAnswers

cpInfoReviewMode

cpQuizInfoPreTestTotalQuestions

cpQuizInfoPreTestMaxScore

cpInfoMobileOS

cpQuizInfoStudentID

cpQuizInfoStudentName

cpQuizHandledAll

DOCUMENTED CAPTIVATE/JAVASCRIPT FUNCTIONS

- <https://helpx.adobe.com/captivate/using/common-js-interface.html>

cpAPIInterface.getVariableValue	Returns the value of the given variable name.
cpAPIInterface.setVariableValue	Sets value of the given variable name
cpAPIInterface.play	Plays the movie.
cpAPIInterface.pause	Pauses the movie.
cpAPIInterface.stop	Stops the movie.
cpAPIInterface.rewind	Rewinds and plays the movie.
cpAPIInterface.next	Seeks the movie to the next slide.
cpAPIInterface.previous	Seeks the movie to the previous slide.
cpAPIInterface.fastForward	Increases the movie speed to 2x, then 4x and then back to normal on consecutive calls.
cpAPIInterface.getPlaySpeed	Returns movie playback speed in Frames per second (fps).
cpAPIInterface.getDurationInFrames	Returns the total number of frames in the movie.
cpAPIInterface.getDurationInSeconds	Returns the total duration of the movie in seconds.
cpAPIInterface.getVolume	Returns the volume of the movie in percentage.
cpAPIInterface.setVolume	Sets the volume of the movie.
cpAPIInterface.navigateToTime	Seeks to a particular time (milliseconds) in the movie.
cpAPIInterface.canNavigateToTime	Returns a boolean value showing whether you can seek to a particular time in the movie or not.
cpAPIInterface.getCurrentFrame	Returns the current frame of the movie.
cpAPIInterface.getCurrentSlideIndex	Returns the current slide index of the movie.
cpAPIInterface.getEventEmitter	Returns the handle to the cpAPIEventEmitter object.

DOCUMENTED CAPTIVATE/JAVASCRIPT EVENTS

- <https://helpx.adobe.com/captivate/using/common-js-interface.html>

cpAPIEventEmitter.addListener (event, function)
cpAPIEventEmitter.removeListener(event)
CPAPI_SLIDEENTER
CPAPI_SLIDEEXIT
CPAPI_STARTPLAYBARSCRUBBING
CPAPI_ENDPLAYBARSCRUBBING
CPAPI_INTERACTIVEITEMSUBMIT
CPAPI_MOVIEPAUSE
CPAPI_MOVIERESUME
CPAPI_MOVIESTART
CPAPI_MOVIESTOP
CPAPI_QUESTIONSkip

FAR TOO MUCH TO “FIGURE OUT” IN CPM.JS

What is an efficient custom interaction development strategy?

- Build basic shapes and simple interactions that do not require advanced actions directly in Captivate
- Use Adobe Documented JavaScript library as starting place
- Developers familiar with HTML/CSS/JAVASCRIPT:
 - Build custom interactions decoupled from the Captivate data structures as much as possible
 - Bridge back into Captivate using the CPM.js library functions
 - Leverage undocumented features only as needed

JAVASCRIPT TO CAPTVATE BRIDGE

1. Variable in captivate are global JavaScript variables

2. Shape names are used to build HTML IDs

div id=re-theSquarec

canvas id=theSquarec

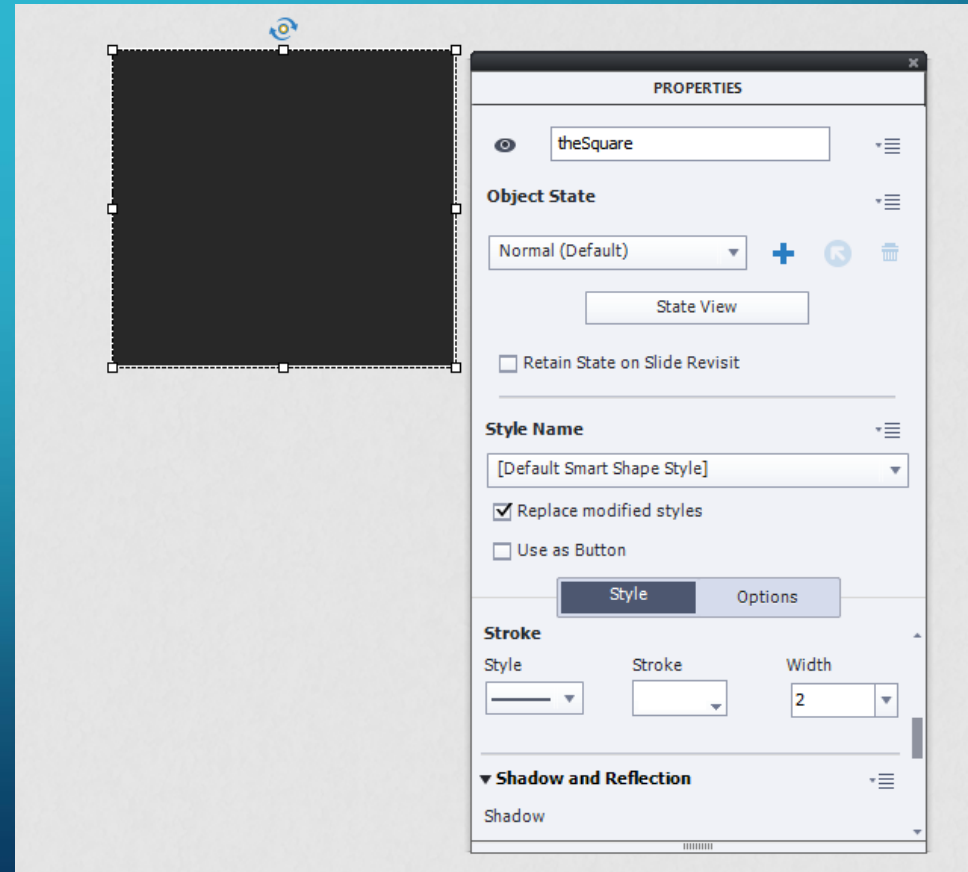
div id=theSquare_vTxtHolder

div id=theSquare

div id=theSquare_vTxtHandlerHolder

div id=theSquareaccStr

3. Use these objects to create custom CSS effects



JAVASCRIPT TO CAPTVATE BRIDGE

- Shape information is found in the object CP.D
 - `cp.D.shapename`
 - `cp.D.shapenamed`
 - `cp.D.shapenamed0`
 - `..others`
- Initiate JavaScript interactions
 - Using button actions in captivate
 - Using JavaScript events tied to HTML objects
- Captivate monitors all variable values once every frame interval (1/30 sec)
 - Simply setting timing-control variables to “true” will cause changes in state
 - Example:
`cpCmdNextSlide = 1`
- Quiz management has another data structure, too much to describe here

Notepad++ JavaScript formatter



```

1  if (!window.cp) {
2      window.cp = function (str) {
3          return document.getElementById(str)
4      };
5  }
6  cp.CPPProjInit = function () {
7      if (cp && cp.model && cp.model.data)
8          return;
9      cp.model = {};
10     cp.poolResources = {};
11     cp.D = cp.model.data = {
12         pref: {
13             acc: 1,
14             rkt: 0,
15             hsr: 1
16         },
17         SmartShape_9: {
18             type: 612,
19             from: 1,
20             to: 90,
21             rp: 0,
22             rpa: 0,
23             mdi: 'SmartShape_9c',
24             retainState: false,
25             immo: false,
26             apsn: 'Slide5611',
27             JSONTT_4: [],
28             cpa: true,
29             oca: 'cp.jumpToNextSlide();',
30             JSONTT_5: [],
31             ofa: 'cp.CmndResume = 1;',
32             vt: '<div><div style="margin-left:0px;display:block;text-align:center">T</span></span><span class="cp-actualText">
33             rplm: {
34                 414: 0,
35                 667: 0,
36                 768: 0,
37                 896: 0,
38                 1024: 0
39             },
40             rprm: {
41                 414: 0,
42                 667: 0,
43                 768: 0,
44                 896: 0,
45                 1024: 0
46             },
47         },
48     },
49 };
50 
```

QUESTIONS?