Whitty Review 12.30.19

This is after the reboot and hard reset to 08d7185, All clipboard buttons work. //cara/arnie still has the Whitty version I was working on with the mobile swap, but that was way too complicated and didn't even work. So we're reverting to the last knowing working version before I started developing that failed mobile swap thing, using the code in scripts.js on /cara/whitty.

IMPORTANT: VS Code is showing a Quick Fix message in all Arnie functions saying 'This constructor function may be converted into a class declaration. It's complicated…probably no good can come of it.

Whitty Code Structure

UIController Functions

Test IIFE to write clicked element to console - commented out.

Alias assignments:

- **dimViewers** - elements in the image data display area: filename, display, diskimg, aspect, smallphones, largephones, retina, clipboardBut. *VA! Bad nomenclature, 'viewer' is used in too many contexts in this app. Change this to 'Inspector Panel', and change each individual dimViewer to Inspector.*
- **toolButtons** - elements in the toolButton area: toolButs, imgDesktop, swapArrow, imgMobile, viewerW, grow50, grow10, grow01, imgWidth, toggleImgSize, imgHeight, shrink01, shrink10, shrink50, sPhonesW, lPhonesW. *VA! Bad nomenclature, should be a distinction between the toolButton area and the individual toolButtons, toolBar should be used for the former.*
- **dynamicRegions** - any element whose size or properties can change as a result of user input or image selection by user: curImg, imgViewer, imgViewport, appContainer. Some of these aliasses are not reflective of the actual HTML elements; this should be revisitied before proceeding. Also, the container relationships of the dynamic regions are complex. Maybe the whole thing could be simplified by rewriting the CSS in flexbox or CSS grid, that's something to keep in mind during this review. *VA! Questionable nomenclature, these container names are all over the place.*
- **staticRegions** - any element whose size or properties don't change based on user input or image selection: dropArea, toolsContainer, ccpContainer, appMessContainer, appMessDisplay, ccpBlocker. *VA! Questionable nomenclature, these container names are all over the place.*
- **ccpUserInput** - CCP elements that take a user input: imgClass, imgAlign, imgRelPath, tdClass, tdAlign, tdValign, tdBgcolor: tdBgimage, tableClass, tableAlign, tableWidth, tableBgcolor, tableIncludeWrapper, tableWrapperClass, tableWrapperWidth, table WrapperAlign, tableWrapperBgColor. *VA! note that dimViewers.clipboardBut is listed here in addition to under the dimViewers, is this an error?*
- **ccpMakeClipBut** - CCP elements that initiate a clipboard action: ccpImgWriteHTMLToCB, ccpTdWriteHTMLToCB, ccpTableWriteHTMLToCB, imgDisplayWriteCSSToCB,

imgSPhoneWriteCSSToCB, imgLPhoneWriteCSSToCB, imgLPhoneWriteCSSToCB, tdDisplayWriteCSSToCB, tdSPhoneWriteCSSToCB, tdLPhoneWriteCSSToCB, tableDisplayWriteCSSToCB, tableSPhoneWriteCSSToCB, tableSPhoneWriteCSSToCB. *VA! Questionable nomenclature, these container names are not consistent or semantic.*

Private Functions

- **evalDimAlerts**: evaluates the width values in the current dimViewers to see if they meet the requirements for retina, i.e. 2X the current width

  Called from: writeDimViewers (public)

  i. Calls UIController.getDimViewerIDs to get the dimViewer aliases to dimViewers.
  ii. Calls appController.initGetAppobj to get imgW, imgNW, sPhonesW, and lPhonesW to props
  iii. Runs UIController.writeDimAlerts to resets all the dimViewer alerts
  iv. Runs UIController.writeDimAlerts again to write the dimAlerts for the current image.
- **clearDimViewers**: Resets the dimViewer Panel. VA! This is in Arnie, but not here. Investigate...
- **resizeContainers(viewerH, imgW, imgH):** This calculates the mainImgViewer, imgViewport and appContainer height based on Appobj values which are passed in from calcViewerSizes. VA! Moved in Arnie to here, not sure why, investigate.

  Called from: UIController.calcViewerSize

  Calls: UIController.writeDimViewers

  i. Sets initViewerH to 450
  ii. Gets the curImg from UIController.getCurImgId
  iii. Sets the viewport to viewerH + 145
  iv. Sets the style.height of viewerH, viewportH and appH. To what? Don't know.
  v. Calls writeDimViewers.

Public Functions

- **Alias getters:** getDimViewersIDs, getToolButtonIDs, getDynamicRegionIDs, getStaticRegionIDs, getCcpUserInputIDs, getCcpBuildTagIDs, getCcpMakeClipButIDs
- **initUI:** Initialize the app with the dropArea is displayed.

  Calls: clearDimViewers

- **queryDOMElements**: This gets all the dimViewer elements to an array. Gets computed styles for imgViewer, plus some other quirks. Take a good look at this and compare to how it's done in Arnie.

  Returns: els (all the dimViewer elements)

Called from: getAppdata

- **writeDimViewers:** Writes all the property data obtained from appController.initGetAppobj to the dimViewers, then calls evalDimAlerts.

  Calls: evalDimAlerts

  Called from: showMainimg(obj), doShowSwapImg, resizeContainers,

- **writeDimAlerts:** Applies the style.color red to dimViewers flagged as alerts by evalDimAlerts.

  Called from evalDimAlerts(dimViewers)

- **flashAppMessage:** Flashes a message in the message container, either an error or status message.

  Called from: messageHandler

appController Functions

Alias Getters: dimViewers, dynamicRegions, staticRegions, toolButtons, ccpUserInput, ccpMakeClipBut

Private Functions

- **SetupEventListeners**: sets up event listeners. Uses bracket syntax I have to review. and includes the following subroutines
  - *runMe*(): Runs function on DOMContentLoaded
  - *dropZone*: can be folded into other event handler routines.
  - *tbClickables*: all toolbar buttons
  - *tbKeypresses*: keydown, keyup, focus and blur events for all the toolbuttons that take user input
  - *ccpKeypresses*: event handlers for the input elements that show mobile CSS clipboard buttons in the CCP when input is made. Runs showElementOnInput. VA! Check to make sure all of them work!
  - *dvClickables*: click handlers for dimViewers. Only the clipboard button is active now, I think, but there could be more.
  - ccpMakeClipBut: runs doClipboard(). VA! Not sure exactly what this does, investigate.
- **handleFileSelect(evt):** Gets the user selected image file object. This uses a setTimeOut to wait while the image data is loaded before running an onload function to do DOM manipulation. Then it initializes the data attributes for the sphones and lphones width using setAttribute and getAttribute. Then it runs calcViewerSize. Then, there’s a callback function that writes the image to the DOM once the image is loaded. VA! Review this, I’m not sure why I did it this way.
- **handleDragOver(evt):** Prevent the bubbling of the event to the parent event handler.

- **handleFocus(evt):** Select the clicked element's value or if no value, set it to empty with cursor.
- **handleBlur(evt):** If blurring from imgW or imgH, clear the field to display the placeholders. Otherwise, restore the field value to the Appdata property !VA NOTE: This can probably replace the blur statements in handleKeydown
- **handleMouseEvents(evt):** Preprocess mouse events and route them to respective eval handler. There's a click event type handler, and then a switch that only has one case. That's not good. VA! I think this was improved on in Arnie, investigate.
- **handleKeyDown(evt)**: This is where the non-default input handling is done for the enter and tab keys. It's explained in the comments. Runs evalToolbarInput. VA! Compare with Arnie to see if there were any improvements.
- **handleKeyUp(evt)**: keyPress handler for the ESC key. This has to be handled on keyup, so we need a separate handler for it.
- **checkUserInput(args)**: Parse keyboard input based on Appdata property passed in from handleKeyup. This is where we identify user input errors if the input isn't allowable based on container size. VA! Compare with Arnie to see if there were improvements. Also, check the argument 'args'. I think I might need to standardize auf 'props', if that appears elsewhere in the same context.

  Arguments:

  Called from: handleMouseEvents, handleKeydown

  Returns: isErr

- **evalToolbarInput(args)**: This processes the user input based on which toolbar input field it was entered into, I.e. height, width, viewerW etc.

  Called from: evalToolbarInput

  Runs: updateAppdata, calcViewerSize.

- **updateAppdata(params...):** Each param pair is a property name prop and a value val. evalToolbarInput passes in one or more such pairs whose corresponding Appdata DOM element/data attribute has to be updated. So, loop through the argument arrays and update the corresponding DOM elements. I think I used params... because not all of the parameters have two arguments.

  Called from: evalToolbarInput

  Calls: initGetAppdata

- **calcViewerSize:** This calculates the viewer size based on the Appdata properties from appController. initGetAppdata. Uses a case select to parse through the potential image size cases. VA! The comments say: PROBLEM: this is only good for initializing because it calculates the viewer size based on NW and NH. On user input, it has to calculate based on imgW and imgH, so compare with Arnie to see if I came up with a more comprehensive solution.

Calls: resizeContainers.

Called from: initDev, handleFileSelect, evalToolbarInput

- **resizeContainers(viewerH, imgW, imgH)**: This calculates the imgViewer, imgViewport and appContainer height based on Appdata values which are passed in from resizeContainers. Initial height is 450, as it is defined in the CSS. TOo much hassle to try and get the value as defined in the CSS programmatically.
- **initCCP()**: Calls initGetAppdata, toggles ccpContainer.active, and handles all the CCP checkboxes, initialzies all the Wrapper table options as 'none'. VA! See how this is done in Arnie, I remember improvements.

  Called from: initDev, setupEventListeners

- **handleCCPInput(event):** Runs initGetAppdata, shows wrapper items if Include wrapper table is checked, toggles checkboxes as necessary. VA! Compare with Arnie, I don't know what the difference is between this and initCCP.

  Called from: initCCP

- **showElementOnInput(event)**: Here we catch the event handlers for the CCP class input fields and show the mobile clipboard buttons when an input is made. The input event fires whenever an input element's value changes.

  Called from: setupEventListeners/ccpKeypresses

- **appMessages(isErr, messCode)**: Stores for all app error and status messages and their error codes.

  Returns: isErr, mess

  Called from: messageHandler

- **elementIdToAppdataProp(str)**: Gets the Appdata property that corresponds to the ID of the DOM input element that sets it. It's easier to just create a list of these correspondences than to rename the whole UI elements and Appdata properties so they correspond, or to create functions that use string methods to extract them from each other.

  Called from: handleBlur, handleKeydown, handleKeyUp

  Returns: ret

- **getAspectRatio(var1, var2)**: Gets the aspect ratio of the two parameters and returns the ratio as a real number or an integer.

  Returns: aspectReal, aspectInt

  Called from: getAppdata

- **initDev:** Shows toolsContainer and curImg. Parses filename ot of path. Gets and sets swphonesw and lphonesw from data attributes. Sets active on ccpContainer and runs initCCP. VA! There were some big changes to this on Arnie, investigate.

  Calls: initCCP

  Called from: init

- **getAppdata():** gets DOM elements from queryDOMElements, gets aspect and computes sPhonesH and lPhones,

Public Functions

- **return:** VA! This is a pass-thru that exposes access to these functions. VA! I can't remember how this is called or whether I recoded it for Arnie, investigate.
  - **initMessage:** runs messageHandler which returns isErr and errCode.
  - **initGetAppdata:** This just exposes getAppdata to other modules and returns Appdata, somehow, see above..
- **init**: Initializes the app. If the curImg exists in the DOM already, then it's set in the HTML file in Dev mode and initDev is run. Otherwise, it runs UICtrl.initUI. VA! I think this was done differently in Arnie, investigate.

CBController Functions

Private Functions

- **getKeyByValue:** This is not accessed in Whitty, but I think it's used in Arnie – VA! Investigate...
- **ccpGetSnippet(id):** This determines which clipboard button is the click target and then gets the corresponding clipboardStr.

  Calls: all of the get clipboardString routines based on a switch/case statement.

  Called from: doClipboard (CBController public)

  Argument: Appdata. VA! I think this was changed somewhat in Arnie, investigate.

- **ClipboardOutput(classAtt, AlignAtt):** Constructor for the clipboard output objects. These are all the properties all the clipboard output objects (img, td and table) will have. We store these key/value pairs in instances of the ClipboardOutput because they're easier to manage. Then we write the output into an HTML string. VA! Don't understand this, see if it's in Arnie.
- **getImgWriteHTMLTB, getTdWriteHTMLToCB, getTableWriteHTMLToCB, getImgDisplayWriteCSSToCB, getImgSPhoneWriteCSSToCB, getImgLPhoneWriteCSSToCB, getTdDisplayWriteCSSToCB, getTdSPhoneWriteCSSToCB, getTdLPhoneWriteCSSToCB, getTableDisplayWriteCSSToCB, getTableLPhoneWriteCSSToCB:** These are the functions that build the respective clipboard outputs for the respective clipboard buttons.
- **ccpIfNoUserInput(att, value):** Handles what to do if there if there's an entry or no entry in the input eleements, I.e. whether the attribute string should be included in the

clipboard output or not and whether to include the filename without the path if the path field is empty. VA! There's a TODO in the comments to put this in handleUserInput, investigate.

Public Functions

- **return:** passthru functions to expose globally:
    - doClipboard(evt):

      Calls: ccpGetSnippet

Whitty/Arnie Comparison

getAppdata vs getAppobj

getAppdata doesn't do anything but call queryDOMElements and compute sPhonesH and lPhonesH based on getAspectRatio. Then it returns Appdata. queryDOMElements is a public UIController function that gets values from the DOM elements, i.e. the dimViewers. The goal is to use the DOM to store the image data rather than trying to manage the data in persistent arrays.

getAppobj is a much better ES6 implementation. It's called globally from appController initGetAppobj. I can't remember why I didn't just include getAppdata as a UIController public function rather than have the passthru initGetAppobj as an appController public function.

clearDimViewers

This isn't in Whitty but is in Arnie. It should probably be in Whitty. It basically resets the dimViewer display to No Image.

showMainImg, showSwapImg, retIsImg, getSwapObj

Swap-specific functions, not in Whitty.

resizeContainers

This is appController private in Whitty, UIController private in Arnie. Should be UIController.

initUI

This UIController public function resets dimViewers to No Image and is called from init in both Whitty and Arnie, but all it does is run clearDimViewers. Not sure why it's necessary at all – resetting the dimViewers should be done initiated in handleFileSelect and init only, right?

calcViewerSize

This is appController private in Whitty and UIController public in Arnie. In Arnie, initGetAppobj puts all the dimViewer values into props. I think using props is better because you're only calling the properties you need rather than the entire Appobj/Appdata. Plus, props is a local object…but then again so is Appdata because it's initialized in each function. I think I used props and ES6 here because Appobj was getting so cluttered with all the swap data that I didn't want to call the whole object, just wanted to call those values I needed for this particular function.

writeDImViewers
Again, in Arnie we're getting individual dimViewer values into props uising initGetAppobj rather than getting the whole Appdata object. Everything else looks the same and it's UIController public in both Arnie and Whitty.

writeDimAlerts
Exactly the same in Whitty and Arnie.

flashAppMessage
Looks to be exactly the same in Arnie and Whitty

ccpGetSnippet
In Whitty, we get the Appdata here and pass the object to each function that gets the clipboard string. In Arnie, we don't get Appobj data here, but rather in the individual function, where we only get the properties needed by that specific function into the props array. Much better.

ClipboardOutput
Constructor is the same in Whitty and Arnie.

updateAppdata vs updateAppobj
Similar in Whitty and Arnie, except in Whitty it's appController private, and in Arnie it's UIController public. Also, Arnie includes the filename and gets it from the dimViewer HTML element.

setupEventListeners
In Whitty, dropZone is the appContainer. I don't think that's right, it should be just the container that holds the drop area. Aside from that and the fact that Arnie has all the swap elements that Whitty doesn't need, it's the same.

handleFileSelect
The filename is handled a little differently in Arnie. In Whitty, it's writted to the DOM as soon as HFS gets it. In Arnie, it's written to the DOM from initDimViewers. Plus, there's some extra code in Arnie to determine whether it's swapMode or mainMode. Aside from that…the same.

handleFocus
ESLint error – evt is passed in as parameter but never used. The click target access this instead of evt. Plus el isn't explicitly defined.

handleBlur
Same ESLint error as above. Also, Whitty access Appdata here probably unnecessarily.

handleMouseEvents
Same ESLint error as above. Also, there's only one case condition here, that doesn't make much sense. Arnie looks to be much cleaner.

handleKeydown
This is a bit complicated but it expresses non-default field behavior:

- imgW and imgH fields should never show entereed values but rather only placeholders. This is because they actual values are reflected upon entering in the DISPLAY SIZE dimViewers and

because any value entered in one of the fields would require an aspect ratio calculation to display in the other one. So one of the field values would have to update automatically which is distracting and confusing IMO especially since the values are presented clearly elsewhere.
- The other fields should show the current Appdata value, i.e. the actual DOM element dimensions or data property value, because these values are NOT reflected anywhere in a dimViewer. So when they are changed, they need to be updated and when a user makes a bad entry, they have to be restored to what they were previously.
- Default Tab behavior, i.e. cycling through the tab order, has to be maintained under consideration of the above 2 points.

There are ESLint errors, keydown isn't defined.

In Whitty: gets all Appdata properties into Appdata. In Arnie, looks like it gets all Appobj properties into props. Aside from that, the two look identical.

handleKeyup

Whitty: Appdata, Arnie: props

checkUserInput:

Arnie: props, Whitty: Appdata. Arnie bets only the dimViewer properties, Whitty gets them all. Error handling for image width, sphonesW and image height aren't yet implemented.

evalToolbarInput

Arnie uses props instead of Appdata. Bestides that, everything else seems identical.

updateAppdata vs updateAppobj

In Whitty this is appController private. In Arnie, it's UIController public, I think, the label in the comments is wrong. Besides that, they appear to be identical, except at the end Whitty gets Appdata frim initGetAppdata, not sure why, probably just for console logging.

initCCP

Identical in Whitty and Arnie

handleCCPInput

Identical in Whitty and Arnie, except Whitty gets properties to data, Arnie gets them to props.

showElementOnInput

Identical in Whitty and Arnie.

appMessages

Identical in Whitty and Arnie.

- **updateAppobj:** updates Appobj based on the ...params passed in from evalToolbarInput.