# Developing a Quink plugin for deviantArt muro

## 1) Put the code in the Quink plugins folder and launch it manually in its own html page.

Since the plugin will be used inside an iframe, a good first step is to make sure it runs at all.

deviantArt's documentation says that it's expected to be used by placing the sandbox page inside an iframe (https://github.com/deviantART/embedded-deviantART-muro/wiki/How-It-Works), so put the deviantArt muro code files in Quink and test that the sandbox page runs in isolation.

Actual result: created quink/plugins/deviantart-muro folder containing code. deviantart\_muro\_sandbox.html ran successfully.

Check in code.

## 2) Create a skeleton plugin adapter

Refer to the Appendix in Quink-Plugin-Notes. deviantart-muro-plugin.js should contain the API functions open, save and exit (don't think a "load" is needed as this is done as part of running the adapter script). At this stage each function should log to the console so we can see it's being called and then call Context.publish for the appropriate message.

console.log('[' + new Date().toISOString() + ']' + 'DeviantArtPlugin.open called');

Run Quink to check it still works (shouldn't be a problem as no new code is being used yet).

Actual result: Quink ran OK

Check in the code.

## 3) Link the minimal plugin to Quink

a) In file resources/plugins.json check the existing insert-key values. Choose a new key ('d') to launch this plugin and get the keycode for this key (68) – Google 'javascript char code list' to get a lookup table for these)

b) add a '68' object to plugins.json, specifying appropriate values for each key.

c) create file logo.png in folder deviantart-muro

## d) launch Quink and check that the new image logo appears in the toolbar. Click on the new logo and this should cause the following (i) adapter script loaded (ii) fetchPluginArtifacts is called (iii) loaded event published (iv) open event triggered (v) plugin close menu button appears at top left or top right (depending on what was specified in plugins.json)

e) Check the operation of the plugin menu button

|  |  |
| --- | --- |
| Event | Expected result in console (custom messages in skeleton) |
| Click on new plugin's icon | loaded script for: ./quink/pluginadapters/deviantart-muro/deviantart-muro-adapter.js  DeviantArtPlugin.open(data) called |
| Click on menu button, continue | (Nothing logged) |
| Click on menu button, save | DeviantArtPlugin.save() called |
| (reopen plugin and) Click on menu button, exit | DeviantArtPlugin.exit() called |

Actual result: these messages were generated

Check in the code.

## 4) Run the deviantart app inside the iframe at launch

a) in pluginadapters/deviantart-muro, create the css that contains the necessary class to display the iframe. Key features of this CSS are

fixed position

height and width 100%

z-index large enough to bring it to the front.

b) in pluginadapters/deviantart-muro, create the html, deviant-art-muro-embed.html, that contains the necessary iframe to contain deviantArt muro. This will contain the

src of the sandbox html, quink/plugins/deviantart-muro/deviantart\_muro\_sandbox.html.

css class for the iframe

class qk\_invisible, initially

b) in fetchPluginArtifacts() add the code to load the css and html from a) and b) then

create a jQuery object $iframe containing the loaded iframe html

publish the loaded event once those two tasks have completed.

c) in open(data),

append the iframe contents to the body tag

remove class qk\_invisible from $iframe

d) in save() and exit(),

add class qk\_invisible to $iframe

call $iframe.detach() to remove it

With the browser debug window open, check that clicking on the deviantArt button on the toolbar adds the iframe HTML (and displays the plugin) and that clicking Save or Exit remove the iframe HTML.

Actual result : this was the outcome.

Check in the code,

## 5) Refactor code (optional)

This is a good point to do any code refactorings and functionality tidy ups before adding the save functionality.

a) Refactor any chained Ajax callbacks into separate functions to make code more readable.

b) Refactor hard coded tag strings into constants.

## 5) Add save functionality

Clicking the Quink "save" button will result in the save() API function in the plugin adapter being called. The plugin adapter is then responsible for getting the required HTML fragment, tidying up (removing listeners etc.) and then publishing a "saved" event using Context.publish.

Sometimes this can all be done inside the save() function, but with deviantArt, the HTML fragment is obtained by sending deviantArt a message and listening for the response.

So:

a) in save(),add a window event listener for "message", handleQueryImageReply, that will check for a query reply

b) in save(), post a query image message to the iframe content window (deviantArt is listening for this)

c) in handleQueryImageReply, check for the presence of message.data.image (this is the base-64 encoded src of an image) and if present then create an image tag with this src, remove event listeners from window and publish the saved event with the img HTML as the data

Test that the image is saved (actual result – the deviantArt image was saved correctly to the main page)

Check in the code.

## 6) Have double tap reopen the plugin with the saved image loaded for editing

The plugin's open(parm) function is called when the image is double tapped (the link between the saved image and the plugin is established in plugins.json)

If the open function's parm is set, then this contains the HTML fragment that must be passed to deviantArt. As with save, processing can often be completed within the open function, but for deviantArt it is necessary to send deviantArt an importLayer message and listen for the reply. So

a) in open(parm), store the parameter (which is the HTML fragment) in a variable , imageHTML, that is available in the "ready" listener's scope.

b) in handleDeviantArtReady if imageHTML has a value then send the importLayer command to deviantArt. Include the imageHTML in the passed object.

c) Publish the Quink "opened" message.

TODO – check what reply comes back from importLayer

TODO – rename the listeners so the naming is consistent.

TODO – find out how you wait until all the DOM for an iframe is loaded – saves having to do the repeated calling

# ToDo

1) Can set default background using options.background = 'http://somewhere.on.my.domain/fancy\_background.png';

Could incorporate into PluginAdapter

2) Copyright licence on plugins

3) Orientation change