IBM Connections Customizer

IBM Connections Customizer is a middleware proxy service that enables the customization of the IBM Connections UX (User eXperience). In essence Customizer acts as a proxy between IBM Connections and the end-user, which gives it the ability to intercept and modify requests and responses, and thus customize anything that flows through it, e.g. the behaviour of APIs, the presentation of the user interface, etc. This document focuses on customizations of the user interface.

The IBM Connections Customizer model is simple – the service performs customizations by injecting JavaScript, CSS or other web resources into the HTML pages returned by IBM Connections in response to end-user requests, i.e. the requests generated within standard components like Communities, Profiles, Files, Homepage and so on, as an end-user navigates the apps. The customization details (typically what code should be inserted and on what requests) are defined by applications stored inside the IBM Connections Application Registry - aka App Reg.

App Reg is a centralized design repository used to store and retrieve applications that customize and extend a variety of different IBM Connections services – where Customizer is just one such service. In the cloud, App Reg is available to organization administrators via the **Admin > Manage Organization > Organization Extensions** menu path. From here it's possible to create and manage Customizer customization applications, which are simply JSON files containing design information defining the components that need to be targeted and the actions that need to be performed. Listing 1 is a simple illustration of a Customizer application.

Listing 1 – Hello World Customizer App

```
1   {
2    "services": [
3      "Customizer"
4    ],
5    "name": "Simple Customizer Sample",
6    "title": "My First Customizer App",
7    "description": "Perform a modification to the Connections Homepage",
8    "extensions": [
9      {
10       "name": "Hello World Extension",
11       "type": "com.ibm.customizer.ui",
12       "path": "homepage",
13        "payload": {
14          "include-files": [
15            "helloWorld/helloWorld.user.js"
16          ]
17        }
18      }
19    ]
20  }
```

The application JSON in Listing 1 requires little explanation. The following points can be inferred by a quick perusal of the code:

- The app is named "Simple Customizer Sample" and it extends the Customizer service
- It contains **one** extension named "Hello World Extension" (apps can have many)
- The extension is a customization of the UI (line #11 - `com.ibm.customizer.ui`)
- The customization applies to the Connections homepage (line #12 - `homepage`)
- A file named `helloWorld.user.js` is to be injected into the homepage (line #15)

A more complete summary of the properties used in Listing 1 is shown below:

Listing 2 Customizer Application Properties

Property	Description
Name	String used to identify the extension
Title	Short string description - translatable for international audiences
Description	Long string description - translatable for international audiences
Services	The service(s) with which the application is associated

Type	String used to identify the extension point being implemented – required. Valid values are as follows:
	`com.ibm.customizer.ui`
	`com.ibm.customizer.api`

Path	String value used to identify the component to be customized:
	`activities`
	`blogs`
	`communities`
	`files`
	`forums`
	`global *`
	`homepage`
	`mycontacts`
	`news`
	`profiles`
	`search`
	`wikis`
	`*` Unlike the other path values, `global` does not represent a real URL path element but is a keyword meaning match **all** URLs.

Payload	
match: **url**	A regular expression used to provide more fine-grained target resource matching, i.e. beyond the broad match specified in the `path` property
match: **user-name**	String used to identify one or more users as the target for the customization - **not unique** within a given organization
match: **user-id**	String used to identify one or more users as the target for the customization. This property is unique within a given organization
match: **user-email**	String used to identify one or more users based on email address value
include-files	List of files to be inserted into the response for a matched page request

A Closer Look at Customizer Properties

The properties outlined in Listing 2 can be broken down into two categories:

1. Generic App Reg Properties

   Properties defined for *all* App Reg applications across all services

2. Customizer Service Properties

   Properties specific to the Customizer service, i.e. *everything in the **Payload** section*

In terms of the generic properties, App Reg requires that any application specify **name**, **title**, **description, service** and **type** property values. The Application Registry specification does not require the **path** property to be specified when an application is created, but the Customizer service puts it to good use for every request it processes, as will be seen shortly. Ergo, in reality a **path** value is required also if Customizer applications are to work properly.

Of the generic properties outlined in Listing 2, only **type** and **path** merit any further discussion. A **type** value always equates to an extension point defined by a service. At present Customizer only defines two extension points, `com.ibm.customizer.ui` and `com.ibm.customizer.api`. The former is a declaration that a given Customizer extension performs a modification to the IBM Connections UI, and thus will be handled in accordance with a prescribed UI extension pattern – for example any **include-files** specified in the **payload** are always injected into the response document. The latter is reserved for future use – suffice to say that as a middleware proxy Customizer is capable of modifying API behaviours, but that use case is not catered for in the current Customizer release.

Listing 3 – Examples of IBM Connections URLs

```
/* homepage */
//w3-connections.ibm.com/homepage/web/updates/#myStream/imFollowing/all
//w3-connections.ibm.com/homepage/web/updates/#myStream/statusUpdates/all
//w3-connections.ibm.com/homepage/web/updates/#myStream/discover/all
//w3-connections.ibm.com/homepage/web/updates/#atMentions/atMentions
/* communities */
//w3-connections.ibm.com/communities/service/html/ownedcommunities
//w3-connections.ibm.com/communities/service/html/followedcommunities
//w3-connections.ibm.com/communities/service/html/communityinvites
/* files */
//w3-connections.ibm.com/files/app#/pinnedfiles
//w3-connections.ibm.com/files/app#/person/7f37da40-8f0a-1028-938d-
db07163b51b2
/* blogs */
//w3-connections.ibm.com/blogs/roller-ui/allblogs?email=joe_schmoe
//w3-connections.ibm.com/blogs/roller-ui/homepage?lang=en_us
/* wikis */
//w3-connections.ibm.com/wikis/home?lang=en-us#!/mywikis?role=editor
```

The **path** property value typically represents a path element in the IBM Connections request URL, which in most use cases corresponds to a standard IBM Connections component.

Consider the URLs displayed in Listing 3 - these sample URLs follow a clear pattern where the next element after the IBM Connections cloud domain name identifies the Connections component or application associated with the request. The possible values of this element map to the **path** values enumerated in Listing 2, i.e. `homepage`, `communities`, `files`, etc.

It follows that according as http requests flow through Customizer it can query the Application Registry for any extensions relating to a given request URL and reduce the scope of the result set by specifying the particular in-context **path** value. Thus a REST request from Customizer to App Reg for Files customizations would look like this:

*appregistry/api/v2/services/**Customizer**/extensions?**type**=com.ibm.customizer.ui&**path**=files*

… which translates as "get all UI extensions registered for the Customizer service that apply to Files". This should clarify why Customizer extensions must contain both a **type** and **path** value. One caveat to note with regard to the **path** value is the existence of the special `global` key word. This is designed to address the use case where an extension needs to apply to *all* requests and it would be clearly inefficient to have to create an extension for every possible **path** value. For example, should a customer need to display some corporate footer text at the bottom of every page in IBM Connections then a `global` extension would facilitate that.

In response to the request shown above, App Reg returns whatever number of extensions match these criteria, i.e. a single collection of one or more JSON files just like the one shown previously in Listing 1. It is then up to Customizer to parse and apply the design metadata contained in the returned extensions – and that is where the **payload** data comes into play.

Processing Payload Properties

As should now be evident, the generic **path** property provides a coarse means of querying the Application Registry for extensions pertaining to a given IBM Connections component. The optional **match** properties inside the Customizer **payload** provide a further means of fine-tuning the filtering of extensions and essentially deciding whether an extension should be applied to a given URL request or not. All **payload** properties are meaningless to the App Registry – they are always just passed back to the nominated service container (Customizer in this instance) for processing.

Fine Grained URL Matching

The **match url** property takes a regular expression and evaluates it against the current URL. If it matches then the extensions is applied. If no match occurs, the extension is not applied. This is a powerful feature as the following code snippets will demonstrate.

Listing 4 shows a Communities extension that has a fine-grained URL match applied on lines 14 – 16. In this case the extension is only applied if the Communities `followedcommunities` URL is being processed, and so this extension is ignored for other Communities URLs like those shown in Listing 3, i.e. `ownedcommunities`, `communityinvites`, etc.

Listing 4 – Customizer App With URL Matching

```
1   {
2     "services": [
3       "Customizer"
4     ],
5     "name": "Communities Customization",
6     "title": "UI Customization for Communities I Follow",
7     "description": "Sample to modify Connections Communities",
8     "extensions": [
9       {
10        "name": "Followed Communities Customizer",
11        "type": "com.ibm.customizer.ui",
12        "path": "communities",
13         "payload": {
14           "match": {
15                   "url": "followedcommunities"
16           },
17           "include-files": [
18             " flipCard/commListCardsFlipStyle.user.js "
19           ]
20         }
21       }
22     ]
23   }
```

Similarly, the following fragment shows how a single global extension can be applied to Homepage and Communities but nothing else:

Listing 5 – Global Customizer App With URL Matching

```
…
12        "path": "global",
13        "payload": {
14          "match": {
15                  "url": "homepage|communities"
16          },
17          …
```

Note: The design of some IBM Connections components like Homepage are based on the Single Page App paradigm. For example, look at the Homepage URLs at the top of Listing 3 – all contain hashtags which means that new http requests are not fired as the user navigates around the page. Thus Customizer is not notified for example when a user moves from `imfollowing` to `atmentions`. By contrast this is not the case in Communities when a user moves from `ownedcommunities` to `followedcommunities`. Thus a developer can target individual Communities URLs using the **match url** property but cannot use the same technique to match the Homepage hashtag URLs. Instead a `homepage` extension would need to inject a script that would listen for hash change events and respond accordingly. A sample is included in the homepage samples: <u>newsRiverSectionedHashChange.user.js</u> . In particular take a look at the `triggerHashChangeEvent()` function contained therein.

It's easy to envisage many other use cases that would require fine-grained match criteria. For instance, if a customer wants to apply a customization to any Files URL that contains a

GUID then this can be achieved by setting the path value to "`files`" and the match **url** value to "`id=[a-z0-9]{8}-([a-z0-9]{4}-){3}[a-z0-9]{12}`" – refer back to Listing 3 for an example of such a Files URL.

Note: The various braces contained in the regular expression would need to be escaped (i.e. preceded by a backslash character: \) when entered into JSON content stored in App Reg.

Fine Grained Matching based on the Active End-User

The **match** property also accepts various user related conditions based on the current user's name or id. In both cases single or multi-value parameters may be provided, or in JSON parlance a single string value or an array of strings can be specified. The fragment illustrated in Listing 6 shows how a Communities extension can be specifically targeted at specific users based on their user names: Jane Doe and Joe Schmoe in this example:

Listing 6 –Customizer App Targetting Specific Users By Name

```
…
12        "path": "communities",
13         "payload": {
14          "match": {
15            "user-name":[
16               "Jane Doe",
17               "Joe Schmoe"
18             ]
19          },
20 …
```

It is important to realise that user names are not unique within an organization so it's possible to inadvertently target unintended users by employing this technique, i.e. all users of the same name will see the extension. To avoid this scenario it is possible to apply a precise filter by using the **user-id** match property instead, as shown in Listing 7:

Listing 7 –Customizer App Targetting Specific Users By Id

```
…
12        "path": "communities",
13         "payload": {
14          "match": {
15            "user-id":[
16               "20071635",
17               "20071656"
18             ]
19          },
20            …
```

The term "user id" is sometimes referred to as "subscriber id" in the IBM Connection UX and documentation.

The Request Life Cycle for IBM Connections Customizer

To summarize what's been discussed thus far, Customizer is a proxy and all Connections requests and responses flow through it. Customizer queries the App Registry to ascertain if customizations have been registered for components of Connections based on the paths of the URL requests it processes. Whenever App Registry does return application definitions to Customizer, the metadata contained in the JSON payload is used to finally decide whether or not a customization should be applied. This request processing mechanism can be succinctly summarized in Figure 1 as follows:

Figure 1 – IBM Customizer Request Life Cycle
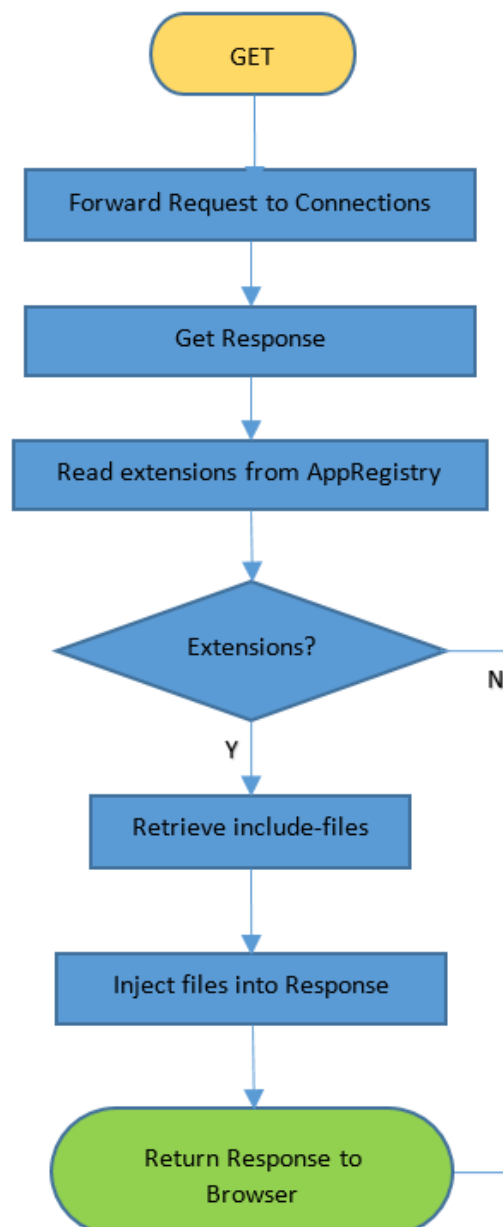
You have already learned how Customizer generates App Registry queries and how request matching is performed based on the application payload data. The next thing to figure out is how the file resources listed in the `include-files` property are managed.

Include Files for Code Injections

The **include-files** payload property lists one or more files to be inserted into the response thus becoming part of the DOM structure loaded in the end-user's browser. Listing 1 shows a simple single-item value for this parameter: `"helloWorld/helloWorld.user.js"`, where `helloWorld` is a folder and `helloWorld.user.js` is a JavaScript file contained within it. This raises a number of interesting questions:

1. **Where do these files reside?**

On IBM Connections Cloud any files declared in the **include-files** property list are stored on an internal persistent volume accessible only by IBM Connections Cloud admin or dev ops staff. They are retrieved for injection as needed by a Customizer microservice at runtime.

2. **How do they get there?**

To make your include-files available on IBM Connections Cloud you need to leverage a web-based source code repository. At this point in time GitHub is the only supported repository though other options may be added in future. If you have a GitHub account then you are already well on your way. If not, then creating a GitHub account is straight-forward and free for public and open-source projects.

Once you have a GitHub account then you should create a source repository to store and manage your Customizer include files. When you are ready to deploy those resources to IBM Connections Cloud then the procedure is as follows:

   a. Share your repo with IBM (https://github.com/ibmcnxdev) – repo invitation

   b. Issue a pull request to IBM when your extension is ready – repo pull request

   c. IBM merges pull request once acceptance criteria are met

   d. Repo files are pushed to IBM Connections Cloud via a webhook upon merge

   e. Rinse & repeat starting at Step (b) for extension updates

The IBM GitHub organization that you need to collaborate with for activities like repository invitations, pull requests, issues and so on is **"IBM Connections Developers"** – refer to Figure 2 for more details.

Step (c) involves a lightweight summary review by IBM which looks at various aspects of the proposed customization, primarily from a performance and security standpoint. However ultimate responsibility for the behaviour of the Customizer application remains that of the customer who creates or adopts the extension. The review process by IBM provides no guarantee whatsoever of protection against adverse security or performance impacts.

Note: If you have a private GitHub repository you can still share it with IBM Connections Developers and it will only be visible to administrators of that organization.

Figure 2 – IBM Connections Developers Organization on GitHub
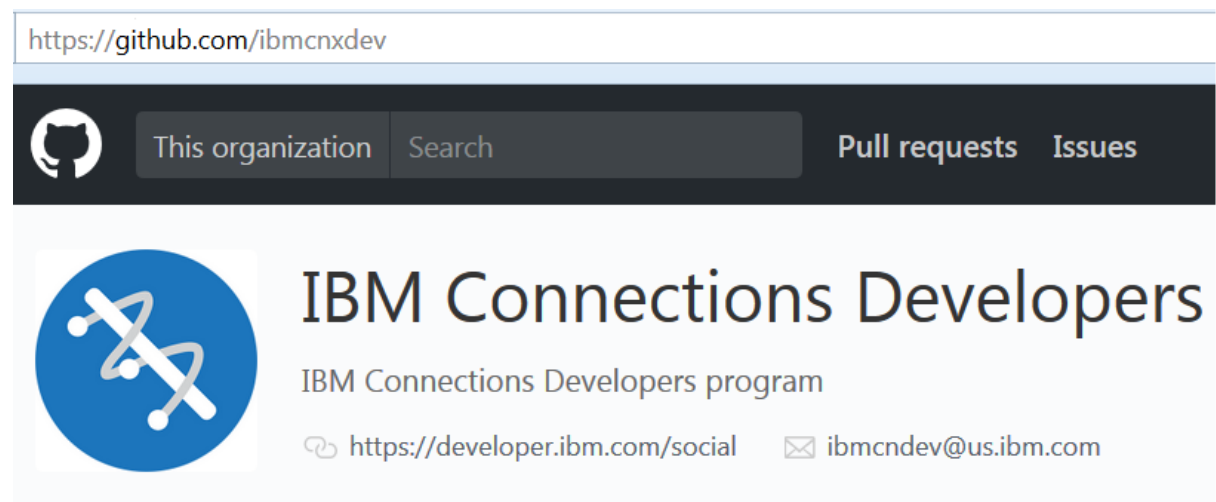
TIP: More information on how to integrate your Customizer include files with IBM Connections Cloud is available in video for on opencode4connections.org:

https://opencode4connections.org/oc4c/customizer.xsp?key=ccc-episode2

Inside the Customizer Include Files

This journey started as most app dev stories do with a reference to a "Hello World" application, the point of which is to jump start the enablement process which the simplest of extensions. So what exactly does the `helloWorld.user.js` include file do? Listing 8 shows the code – certain variable names and comments have been trimmed for readability in this document but nothing that affects the execution of the script.

Listing 8 –Hello World Include File

```
1 if(typeof(dojo) != "undefined") {
2  require(["dojo/domReady!"], function(){
3   try {
4     // utility function to wait for a specific element to load...
5     var waitFor = function(callback, eXpath, eXpathRt, maxIV, waitTime){
6         if(!eXpathRt) var eXpathRt = dojo.body();
7         if(!maxIV) var maxIV = 10000; // intervals before expiring
8         if(!waitTime) var waitTime = 1;  // 1000=1 second
9         if(!eXpath) return;
10        var waitInter = 0;  // current interval
11        var intId = setInterval( function(){
12          if(++waitInter<maxIV && !dojo.query(eXpath,eXpathRt).length)
13            return;
14
15          clearInterval(intId);
16          if( waitInter >= maxIV) {
17            console.log("**** WAITFOR ["+eXpath+"] WATCH EXPIRED!!!
interval "+waitInter+" (max:"+ maxIV +")");
18          } else {
19            console.log("**** WAITFOR ["+eXpath+"] WATCH TRIPPED AT
interval "+waitInter+" (max:"+maxInter+")");
20            callback();
21          }
22        }, waitTime); // end setInterval()
23     }; // end waitFor()
24
25     // here we use waitFor to wait for the
26     // .lotusStreamTopLoading div.loaderMain.lotusHidden element
27     // before we proceed to customize the page...
28     waitFor( function(){
29         // wait until the "loading..." node has been hidden
30         // indicating that we have loaded content.
31
32       dojo.query("span.shareSome-title")[0].textContent="Hello World! ";
33
34     }, ".lotusStreamTopLoading div.loaderMain.lotusHidden");
35
36   } catch(e) {
37       alert("Exception occurred in helloWorld: " + e);
38   }
39  });
40 }
```

For a simple Hello World example, this may appear at first glance to be more complicated than one might expect, but a closer inspection will simplify matters considerably. Before perusing the code be aware of the following points:

- Most of the code in Listing 8 is a re-usable template that any injection code can sit inside

- Just 1 line of code are needed for the actual Hello World UI update: Line 32 - in **bold**

- IBM Connections UI uses the Dojo framework so code is injected into a Dojo structured page

The JavaScript code initially validates that Dojo itself is loaded and then uses a standard Dojo utility (domReady) to wait for the DOM to fully load before calling a bound function to perform the customization. Lines 2 – 23 define a function which will wait up to a maximum of 10 seconds for the page to fully load and if successfully loaded within that time period will execute a callback function. If the page does not load within 10 seconds then an error is logged to the JS console.

Figure 3 Hello World Extension for IBM Connections Homepage
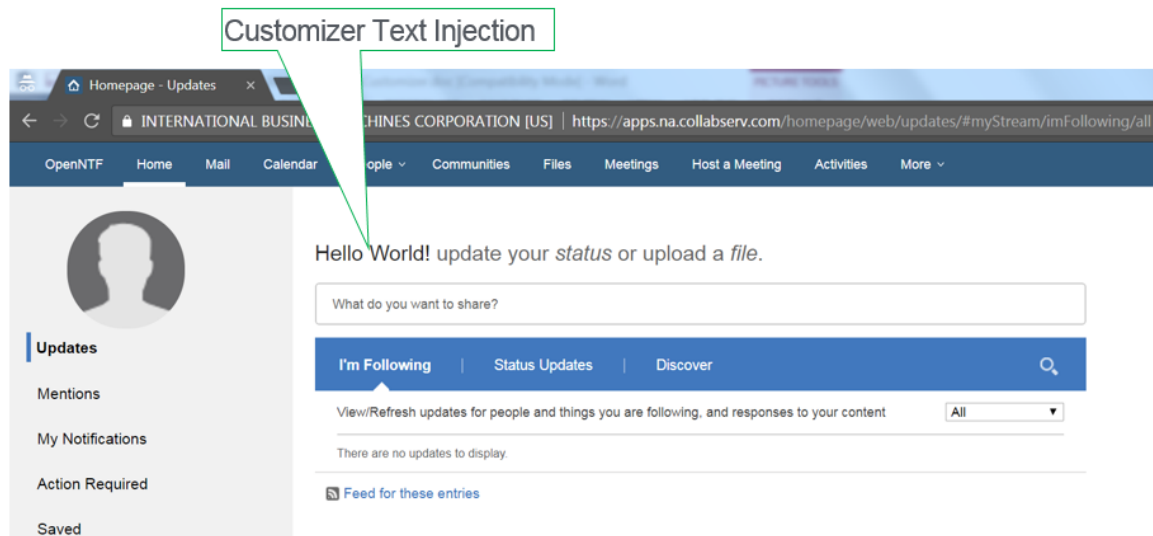
This `waitFor()` function is thus called passing in the callback function to manipulate the DOM and modify the UI. The interesting part of the callback function (Line 31 as already highlighted) locates a DOM element and assigns "Hello World" as the text content. When this extension is loaded and run by Customizer then the IBM Connections Homepage is modified in the manner shown in Figure 3.

The code injection can be seen by viewing the source of the IBM Connections Hompage in the browwswer and scrolling to the bottom. The following tag fragment should be evident (the '/files/muse-static/' prefix in the `src` attribute can be ignored as this is an internal reference):

Listing 9 –Customizer Script Injection

```
…
<script type='text/javascript' src='/files/muse-
static/helloWorld/helloWorld.user.js'> </script>
…
```

TIP: IBM Connections web pages contain a lot of predefined JS variables which can be leveraged by Customizer extensions. For instance, there is an `lconn` (**L**otus **Conn**ections) object with many properties defined that any extension script can exploit. Thus on Line #31, replacing **"Hello World: "** with **"Hello " + *lconn.homepage.userName* + " "** would dynamically include the current user in the Homepage customization. The `lconn` object and others like it should be explored and leveraged by your extensions.

Other Samples

Besides Hello World, there are a number of other ready-made Customizer examples to be available for experimentation. The latest samples can always be found in the samples folder of the Customizer GitHub repository:

https://github.com/ibmcnxdev/customizer

Each sample has its own subfolder which contains the App Reg design definition (JSON file) and the resources to be injected to perform the customization (JavaScript, CSS). Take a look at the following examples:

flipcards

This extension provides an alternative rendering for the Communities pages so that a user's communities can be displayed as flip cards rather than a table of rows. Figure 4 shows a list of three communities with the traditional row based rendering on the left hand side juxtaposed with the flip card layout on the right. Each flip card displays the Communities logo until the user hovers over it whereupon the card is flipped to display the details of the community in question.

## *Figure 4 Communities Page before and after Flipcard Customization*
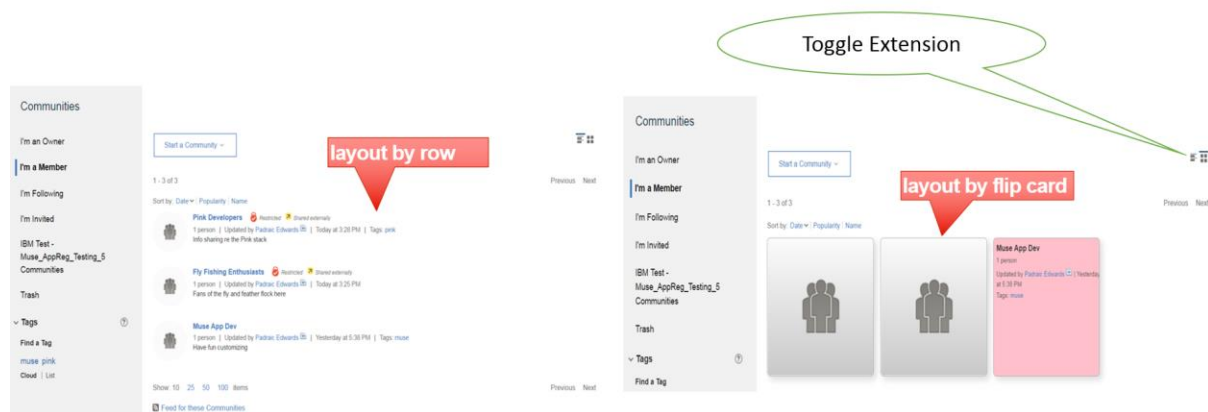
The `flipCard.json` file follows the standard App Reg pattern explained already with the Hello World example. The JavaScript file `commListCardsFlipStyle.user.js` uses the sample Dojo wrapper to envelope the customization but the code itself is significantly more advanced and serves to give a more real-world indication of the art-of-the-possible with Customizer extensions.

Look for the Toggle Extension control on the Communities page when this customization is applied. Clicking the button allows the user to switch back and forth between the standard row layout and the flip card format.

newsRiver

This extension targets the IBM Connections Homepage and reformats the layout of the activity stream updates by accentuating the space surrounding each entry. Figure 5 shows the Homepage when the newsRiver customization is run – note how the entries display as sections against a pink backdrop. Notice that the Hello World extension is also applied to the Homepage? This shows how multiple App Reg extensions can target the same IBM Connections path - viewing the source of the page will show two JavaScript file injections in this case.

## Figure 5 Multiple Extensions for IBM Connections Homepage
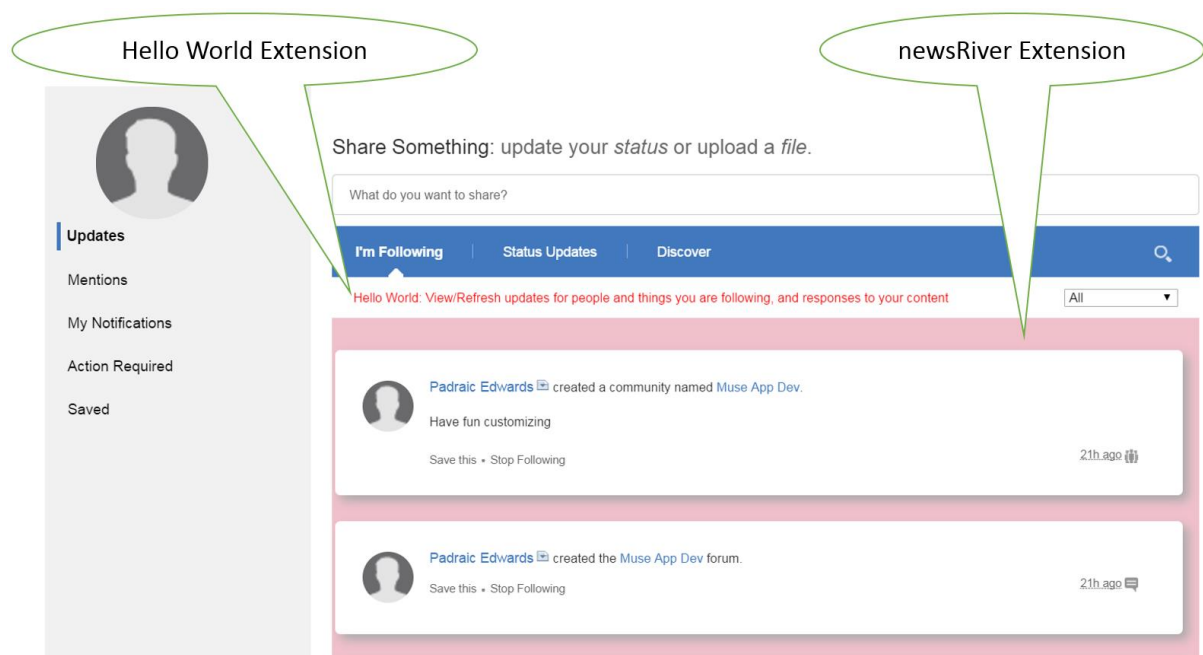
profiles

The Profiles extension delivers a more sophisticated rendering to the page that is displayed when the user selects the "My Profile" dropdown menu option in IBM Connections. The new UI look and feel is achieved via stylesheet updates. There are two files in the `profiles` subfolder - the JS file `profilesCustomization.js` simply inserts a link to the `profilesCustomization.css` file which does all the work. The new look Profiles page is shown in Figure 6.

Note the inclusion of a new page header graphic, the relocation of action buttons and so forth.

Figure 6 Profile Page Extension
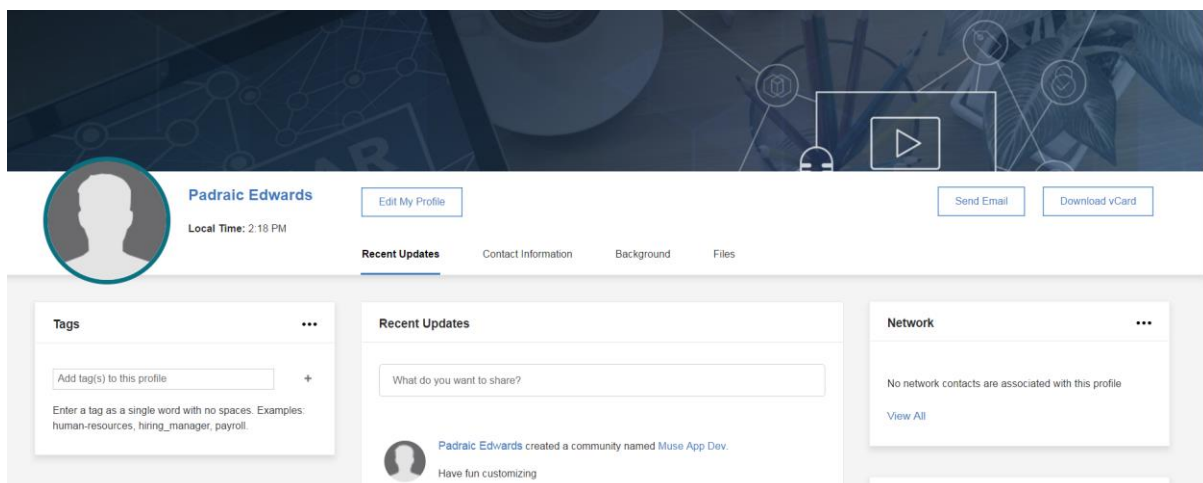
Getting Up and Running

At this time organizations in IBM Connections Cloud are enabled for Customizer by request only. In the future it is envisaged that Customizer will be automatically enabled for all organizations. The manual enablement process is quick and easy however – simply send an email requesting Customizer enablement to `ibmcndev@us.ibm.com` and include your full name, organization name and id. The request is typically completed within 24 hours.

The include-files for all samples discussed in this document are automatically provisioned to any organization that is Customizer-enabled. This means that the samples can be used immediately by importing the relevant JSON file into the organization's Application Registry. So for example, to validate that your organization is Customizer-enabled you could import the `helloWorld.json` file into App Reg as follows:

- Make a local copy of the `helloWorld.json` file

- As org admin user, go to **Admin > Manage Organization > Organization Extensions**

- Click **Add** > **Choose File** on the "**upload an extension from a JSON file"** radio button option

- Select your local copy of the `helloWorld.json`

- Click **Enable Editing**

- Insert a match criterion like the one in Listing 6 so that the extension is only applied to you

    o i.e. match to **your** user-name, e-mail address or user id

- Click **Add** to save the application in App Reg

- Refresh the IBM Connections Homepage and verify that the Hello World extension appears

If this is successful then you have validated that Customizer is properly enabled for your organization by applying a customization that is visible only to you and not all other users. If the customization does not appear then you should report the issue by sending a mail to `ibmcndev@us.ibm.com`. Or reporting an issue on the https://github.com/ibmcnxdev/customizer site.

TIP: The steps outlined above are covered in an enablement video available online here:

> https://opencode4connections.org/oc4c/customizer.xsp?key=ccc-episode1

You can experiment with the other samples in a similar way.

In reviewing the include files you may have noticed that some samples use a JavaScript filename notation that follows the GreaseMonkey naming convention: `somename.user.js.` This is because these customizations were originally developed as GreaseMonkey scripts using browser-based extensions. They were then deployed on the server-side in IBM Connections as Customizer extensions. This option is not only still valid, it is considered a standard practice for developing Customizer apps – i.e. create some new browser extensions using a user script technology like GreaseMonkey for Firefox or TamperMonkey for Chrome. Once you are happy with the local customization then you can submit the resources to IBM for review – i.e. the JavaScript and CSS files you create using GreaseMonkey or TamperMonkey become your Customizer include-files. You can then invoke the customization by creating a Customizer extension in the Application Registry as you have learned here.

Some Points to Note regarding Customizer Applications

- Support for Customizer applications follows the same [policy](#) as any other customization to the IBM Connections UI – i.e. **IBM Support can address questions about the customization process, but cannot address questions about the particulars of your customization**.

- Listing 2 provides the list of supported paths for Customizer at this point in time. This list currently encompasses all core IBM Connections apps but does not include the Administration URLs (aka BSS), or related IBM ICS components like IBM Docs, Meetings or Verse. The list may be expanded to include these and possibly other components in the future.

- Customizer extensions are currently restricted to the organization in which they have been added. For example, users from one organization may have access to communities in other organizations if they have been so invited, but they would not see any customizations added to such "external" communities.

Useful Online References

User Script Technologies:

https://greasyfork.org/en

https://tampermonkey.net/

http://www.greasespot.net/

https://zach-adams.com/2014/05/best-userscripts-tampermonkey-greasemonkey/

https://www.lifewire.com/top-greasemonkey-tampermonkey-user-scripts-4134335

IBM Connections Customizer:

https://opencode4connections.org/

https://github.com/ibmcnxdev/customizer