

Beezy SDK

User Manual

18/02/2021



Contents

Introduction.....	3
Environment setup.....	4
Node.js	4
Npm	4
Mapping a SharePoint site to a local network drive	4
Prerequisites.....	4
Map SharePoint site with local network drive	7
Creating a branding project	9
Yeoman	9
Generating the branding package	9
Deploying the package	13
Development process	13
Deploying changes to SharePoint.....	13
Install project dependencies	13
Build the branding solution.....	13
Generating the branding package	14
Add-ins.....	14
WSP (Full trust code).....	16
Developing a branding package.....	17
Project structure	17
Branding entry point.....	17
Stylesheets	18
Webfonts.....	18
Images.....	19
Colors.....	19
Icons.....	19
Example of an icon creation	20

Variables and mixins.	22
Masterpages.....	23
Multibranding	24
Changes on the Default's Entity branding	24
Changes in masterpage	24
Custom logo for each local entity	24
Changes on the Local Entities' branding	25
Custom Logo	25
Branding.json.....	26
Production.config & debug.config	26
Provisioning package.....	27
Troubleshooting.....	29
Out of memory in build process	29
I changed files but they are not reflected in my Beezy environment	29
ENOENT errors in build process	29
Installation of the branding package fails	29

Introduction

This document will explain all you need to know to create a Beezy branding packages.

It is divided into three main sections:

- Software prerequisites and environment set up
- Creation of a new branding package
- Developing and maintaining a branding package

Note: Changes done in branding won't be reflected in the mobile app

Environment setup

Node.js

Install Node.js 6.10.0 or higher. You can find the latest release [here](#).

Npm

This is one of the most popular package managers for JavaScript so let's install the latest version.

Open Command Prompt and execute **npm i -g npm@latest** :

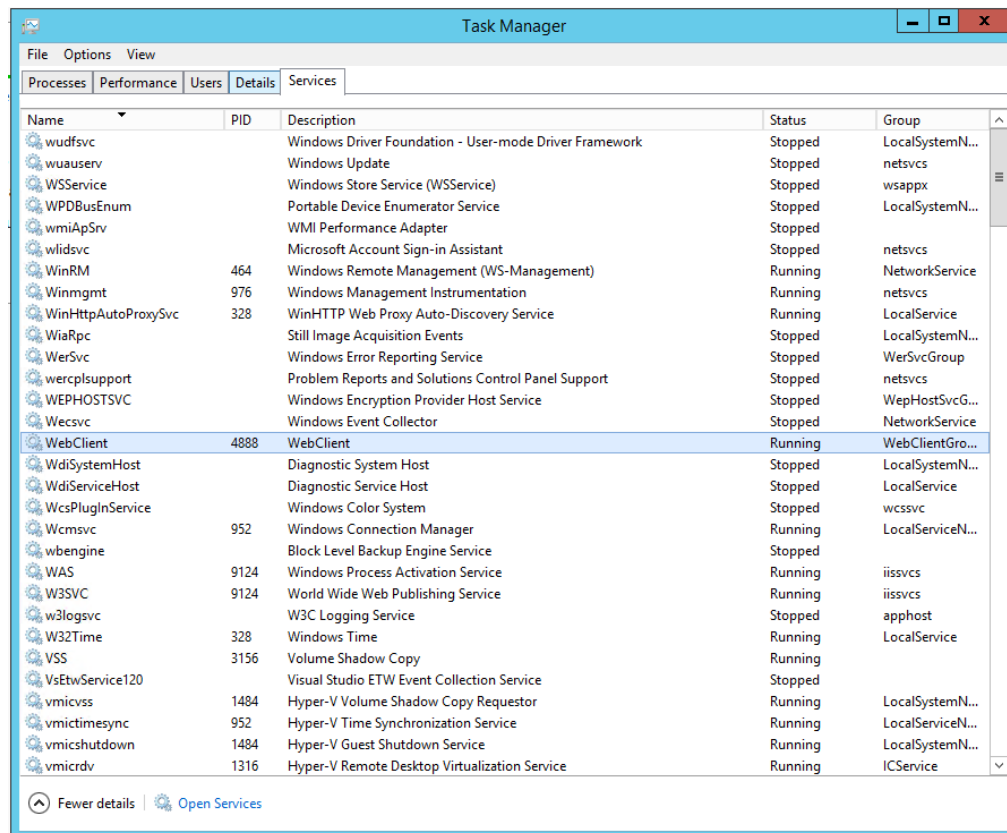
```
PS C:\> npm i -g npm@latest
```

Mapping a SharePoint site to a local network drive

While this requirement is not mandatory, it is very practical to set up a local network drive pointing to the SharePoint site on which to test the branding packages. This way the changes will be reflected in a real environment as they are being developed. A later section of this document will describe how to develop a branding package with a *watcher* to capture all changes as they are saved locally.

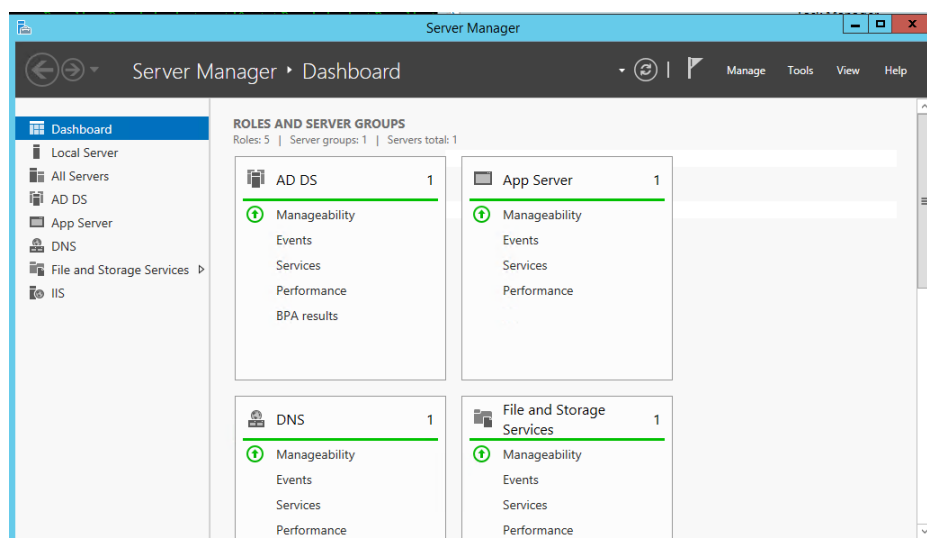
Prerequisites

We need to make sure the "WebClient" service is up and running in the machine where the branding package will be developed.

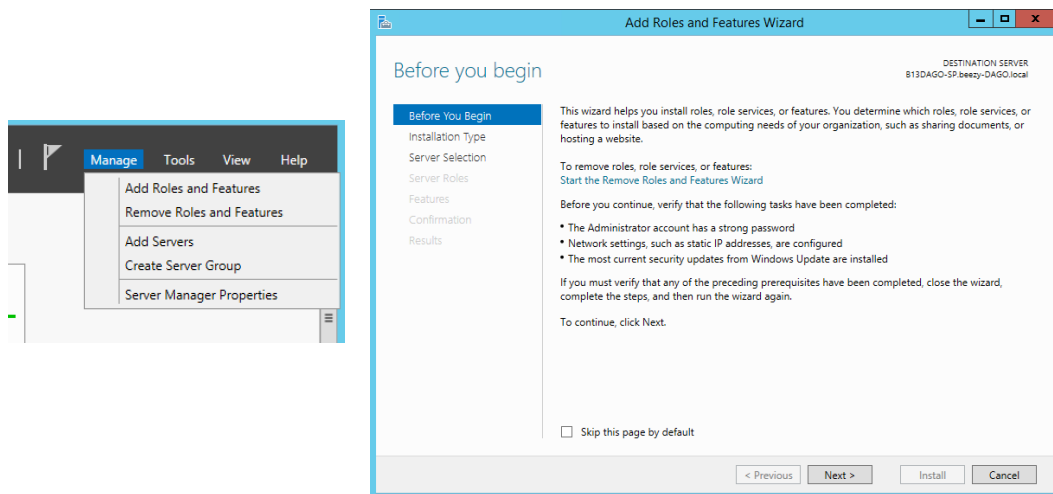


To enable the service please follow these steps:

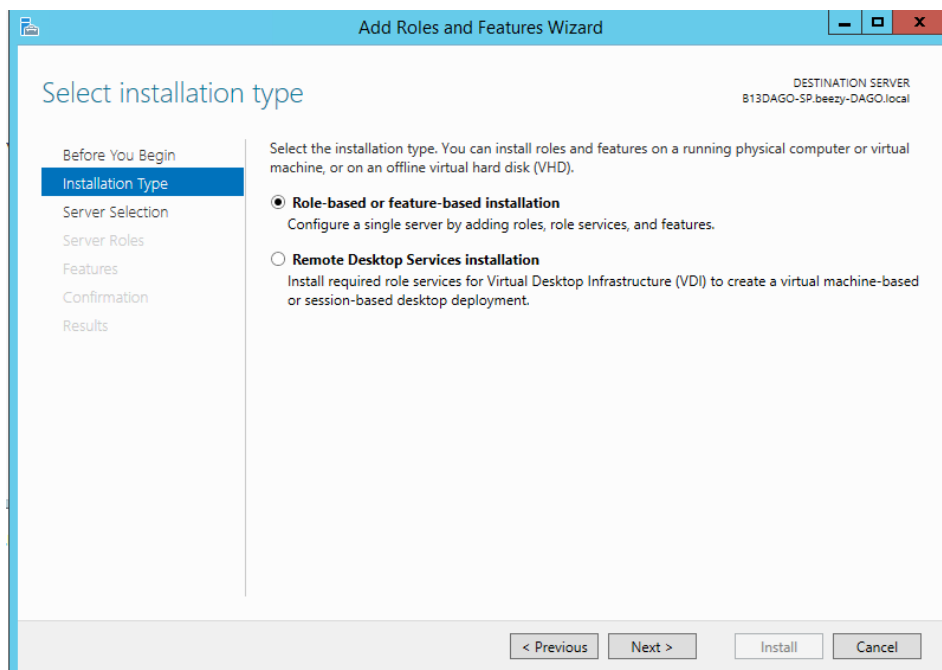
1. Start Server Manager



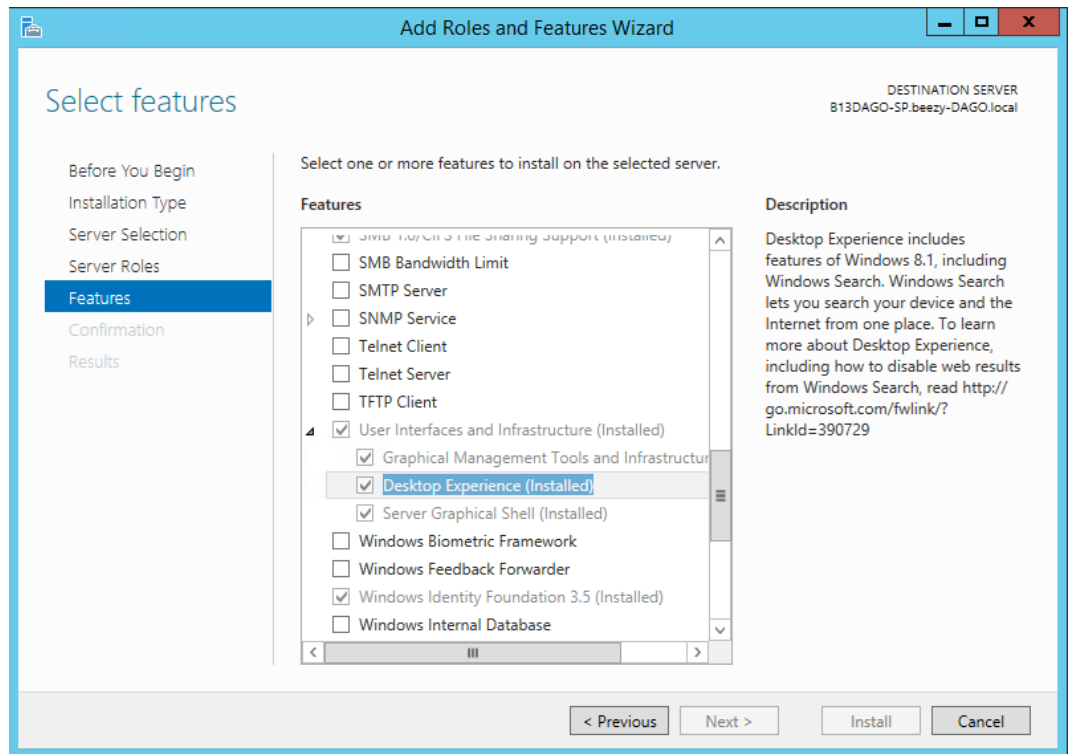
2. Go to *Manage* and click on *Add Roles and Features*:



3. Select *Role-based or feature-based* installation



4. Under *Features*, open *User Interface and Infrastructure* and enable *Desktop Experience*.

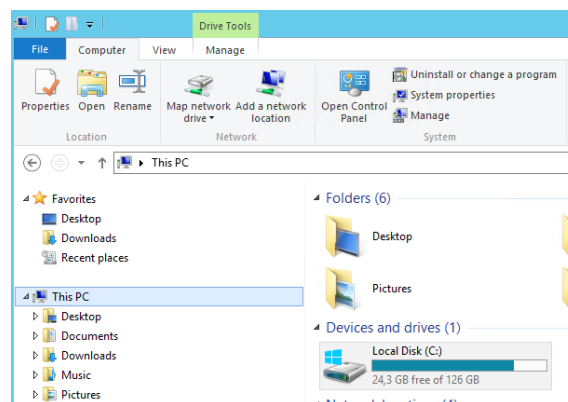


5. Click on *Next* and *Install*.
6. Click "Yes" when promoted to restart the computer.

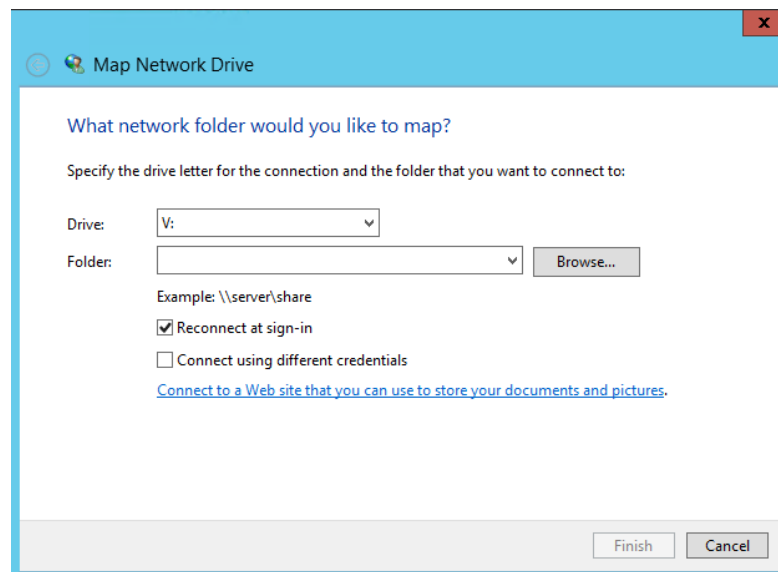
After these steps, you should be able to see the "WebClient" service is Running in the Services window.

Map SharePoint site with local network drive

1. Go to Windows explorer and to This PC.
2. In the top bar, click on *Map network drive*:



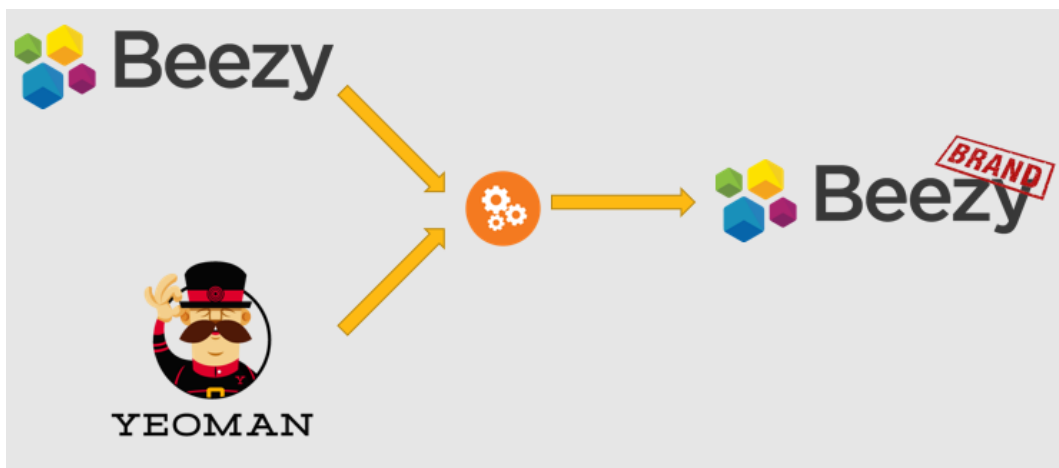
3. In the following window, you can select a name to your network folder. Type in your SharePoint site route where Beezy is installed.



Creating a branding project

Yeoman

Yeoman is a tool that generates all the scaffolding files and artifacts needed to develop a Beezy branding package. With Yeoman, the Beezy SDK will automatically generate an empty branding package ready to be customized in just a few steps.

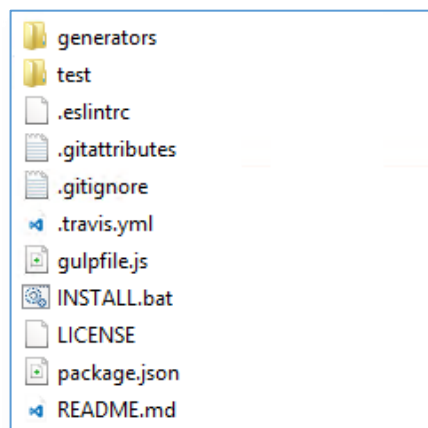


Generating the branding package

Download the latest Beezy SDK from the Beezy SDK Portal:

<https://sdk.beezy.net>

Extract it to a folder of your choice. You should see a similar folder structure than this:



Go to the folder where you extract it, and execute *Install.bat* to install all the Beezy SDK dependencies:

```
C:\BeezyCode\BeezySDK>INSTALL.bat
npm WARN deprecated npmconf@2.1.2: this package has been reintegrated into npm and is now out of date with respect to npm
C:\Users\xare\AppData\Roaming\npm\yo-complete -> C:\Users\xare\AppData\Roaming\npm\node_modules\yo\lib\completion\index.js
C:\Users\xare\AppData\Roaming\npm\yo -> C:\Users\xare\AppData\Roaming\npm\node_modules\yo\lib\cli.js

> yo@1.8.5 postinstall C:\Users\xare\AppData\Roaming\npm\node_modules\yo
> yodoctor

Yeoman Doctor
Running sanity checks on your system

✓ Global configuration file is valid
✓ NODE_PATH matches the npm root
✓ Node.js version
✓ No .bowerrc file in home directory
✓ No .yo-rc.json file in home directory
✓ npm version

Everything looks all right!
C:\Users\xare\AppData\Roaming\npm
-- yo@1.8.5
```

Execute the Yeoman command to start the new Beezy branding generation process:

yo beezy-branding

```
C:\BeezyCode\BeezySDK>yo beezy-branding

  [  (o)  ]
  [  ~  ]
  [  A  ]
  [  ~  ]
  [  o  ]

Welcome to the Beezy branding generator!

? Your customer name (no spaces, e.g: Contoso, Fabrikam, ...): Partner
? For what type of environment are you developing? Addin
? Write the path where the solution will be generated: C:\BeezyCode
? Do you want to use a mapped network drive to update SharePoint with your last changes? Yes
? Enter your mapped network drive: Z:
? Please, enter your style library relative path (e.g: Style Library/Beezy): Style Library/Beezy
? Do you want to configure the Style Library to remove approvals on file changes? (recommended if you are mapped network drive) Yes
? Enter your site collection url: http://portal.b13ciapp1.local/
Partner
Addin
C:\BeezyCode
```

The process will ask a set of questions to configure the branding package.

First, write the costumer name:

```
C:\BeezyCode>yo beezy-branding

  _ _ _ _ _
  | --(o)-- |
  |   'U'   |
  |   A   |
  |   ~   |
  |   o   |
  |   Y   |
  |_____|

  Welcome to the Beezy
  branding generator!

? Your customer name (no spaces, e.g: Contoso, Fabrikam, ...): (BeezyCode) Partner
```

Next, select the environment type (provider hosted add-ins or full trust code):

```
? For what type of environment are you developing? (Use arrow keys)
> Addin
  Onprem Classic
```

Then it asks for the path where the generator will put the new project solution. It is very important that the path **does not contain spaces**:

```
? Write the path where the solution will be generated: C:\BeezyCode
```

The next questions will depend on whether we want to use the mapped network drive that we set up in the previous section of this document. If we want to see the latest changes being deployed immediately to the SharePoint site via the mapped network drive, we must answer yes and type in the network drive letter.

```
? Do you want to use a mapped network drive to update SharePoint with your last changes? Yes
? Enter your mapped network drive: (Y:) Z:
```

Write the relative path of your Style Library inside of Beezy site:

```
? Please, enter your style library relative path (e.g: Style Library/Beezy): (Style Library/Beezy)
```

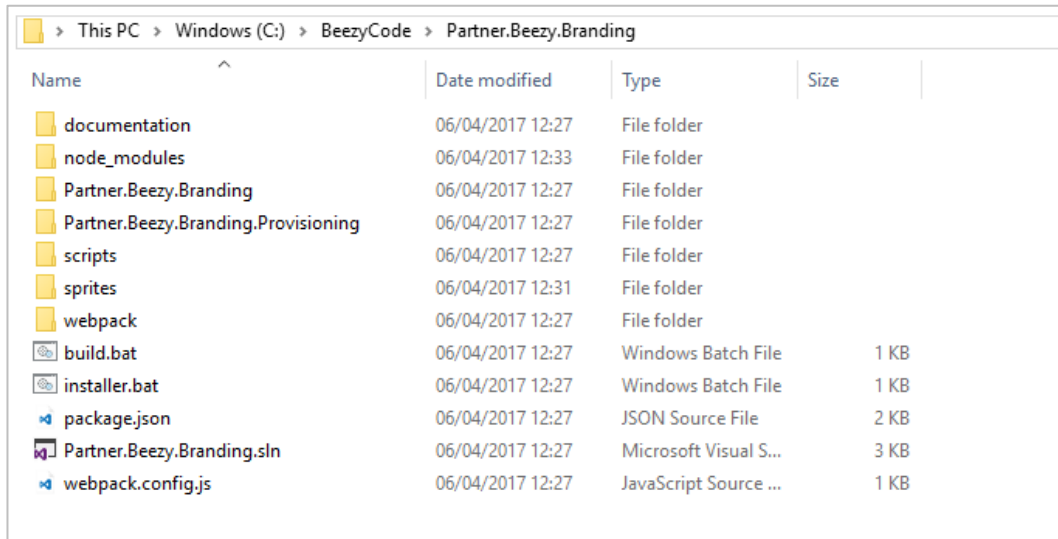
When we are modifying files inside SharePoint libraries, these have the *Version history* feature activated by default, and we have to Check-in any files we modify. The next step of the yeoman generator will allow us to disable this approval of changes so that we can see the changes immediately in the environment.

```
? Do you want to configure the Style Library to remove approvals on file changes? (recommended if
you are mapped network drive) (y/N) y
```

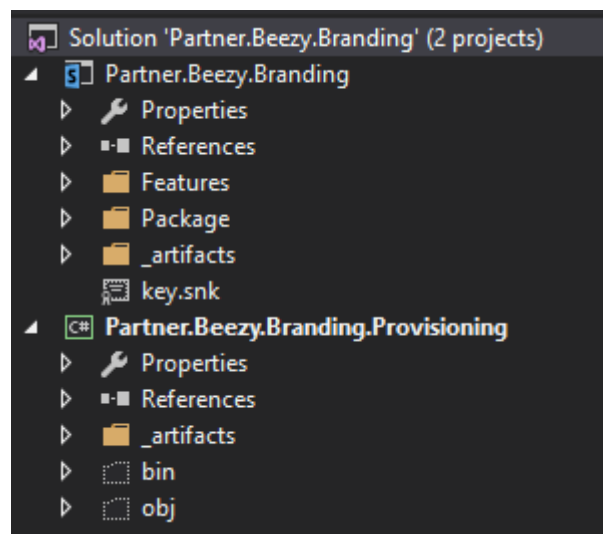
Next we have to write our Beezy site collection:

```
? Enter your site collection url: http://partner.com/sites/beezy
```

After the process finishes, we should have the new branding solution ready. Go to the folder we put above and we can see something like this:



The Visual Studio solution includes be two projects:



The first project contains the source code and elements of the Beezy branding package. This is the project where we'll prepare the branding (customize JS, SCSS, add images, add web fonts, etc.). The second project is used to generate the DLL that will be used to deploy the new branding if we're working on an Add-in scenario. We'll just have to include and embed the auto-generated branding files in this project.

Deploying the package

Development process

It is recommended the use of a watcher when developing a Beezy branding package. The watcher automatically detects changes in the source files and copies the output generated from the build to a folder or to SharePoint directly.

The SharePoint folder should be configured as a network drive (see the **Environment setup** section).

Deploying changes to SharePoint

Install project dependencies

It is very easy to see your changes in SharePoint if you configured your environment to work with a watcher. First, open a Command Prompt, navigate to the root of your Branding package folder and run **installer.bat** to install all the project dependencies.

Build the branding solution

Once the dependencies have been installed, it is time to build the solution. In order to build the branding package you have to execute the following command at the root of your branding package:

build.bat [--watch] [--app=OnPrem|Cloud] --build=release

Parameters:

- **--watch** [optional]: only if you're using *Watcher*
- **--app** [optional]: only if you're in Add-in scenario
 - *OnPrem*: for Add-in on premises
 - *Cloud*: for Add-in Cloud
- **--build=release**: brandings will always be generated in release mode

Generating the branding package

Add-ins

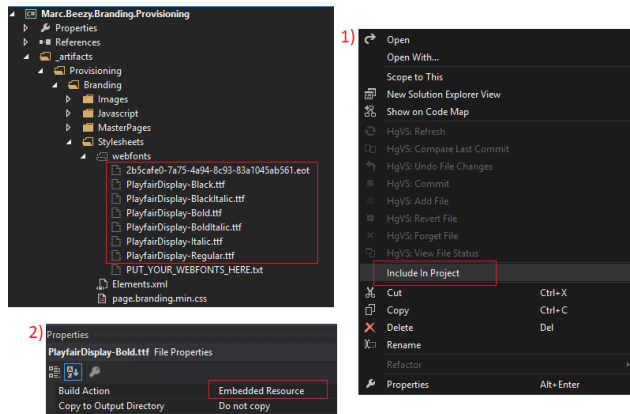
Regardless of whether you are developing using the Add-in model for an on-premises or a cloud solution, the process to generate the branding package is the same.

Previous command **build.bat** will compile JS and SCCS. Then it will copy generated files and all files you've added (web fonts, images, master page, sprite images, etc.) to the provisioning project. You'll see these files in Visual Studio if you select "Show All Files". For all files you've added in the first project that need to be deployed to Beezy you have to:

1. Include the files to the provisioning project
2. Set the "Build Action" as "Embedded Resource"
3. Add the file to the corresponding Elements.xml file with the same format of existing ones.

Example 1 – You've added web fonts. For all files added:

- Select file > Include in project
- Select file > Properties > Build Action: Embedded Resource



- **Elements.xml** should look like:

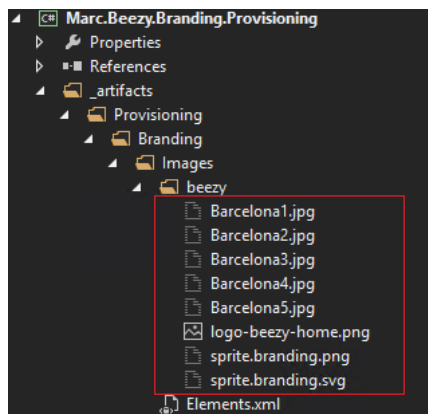
```

1 <?xml version="1.0" encoding="utf-8"?>
2 <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3   <Module Name="Stylesheets" Path="Stylesheets" Url="Style Library/beezy" RootWebOnly="True">
4     <File Path="page.branding.min.css" Url="page.branding.min.css" Type="GhostableInLibrary" />
5     <File Path="webFonts\PlayfairDisplay-Black.ttf" Url="webFonts\PlayfairDisplay-Black.ttf" Type="GhostableInLibrary" />
6     <File Path="webFonts\PlayfairDisplay-BlackItalic.ttf" Url="webFonts\PlayfairDisplay-BlackItalic.ttf" Type="GhostableInLibrary" />
7     <File Path="webFonts\PlayfairDisplay-Bold.ttf" Url="webFonts\PlayfairDisplay-Bold.ttf" Type="GhostableInLibrary" />
8     <File Path="webFonts\PlayfairDisplay-BoldItalic.ttf" Url="webFonts\PlayfairDisplay-BoldItalic.ttf" Type="GhostableInLibrary" />
9     <File Path="webFonts\PlayfairDisplay-Italic.ttf" Url="webFonts\PlayfairDisplay-Italic.ttf" Type="GhostableInLibrary" />
10    <File Path="webFonts\PlayfairDisplay-Regular.ttf" Url="webFonts\PlayfairDisplay-Regular.ttf" Type="GhostableInLibrary" />
11  </Module>
12 </Elements>

```

Example 2 – You’ve added images or custom icons (sprite branding files).
For all files added:

- Select file > Include in project
- Select file > Properties > Build Action: Embedded Resource



- **Elements.xml** should look like:

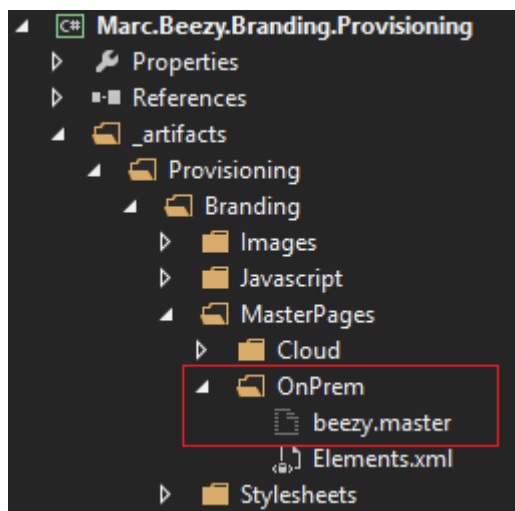
```

1 <?xml version="1.0" encoding="utf-8"?>
2 <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3   <Module Name="Images" Path="Images\beezy" RootWebOnly="True" Url="Style Library/Images/beezy">
4     <File Path="logo-beezy-home.png" Url="logo-beezy-home.png" Type="GhostableInLibrary" />
5     <File Path="Barcelona1.jpg" Url="Barcelona1.jpg" Type="GhostableInLibrary" />
6     <File Path="Barcelona2.jpg" Url="Barcelona2.jpg" Type="GhostableInLibrary" />
7     <File Path="Barcelona3.jpg" Url="Barcelona3.jpg" Type="GhostableInLibrary" />
8     <File Path="Barcelona4.jpg" Url="Barcelona4.jpg" Type="GhostableInLibrary" />
9     <File Path="Barcelona5.jpg" Url="Barcelona5.jpg" Type="GhostableInLibrary" />
10    <File Path="logo-beezy-home.png" Url="logo-beezy-home.png" Type="GhostableInLibrary" />
11    <File Path="sprite.branding.png" Url="sprite.branding.png" Type="GhostableInLibrary" />
12    <File Path="sprite.branding.svg" Url="sprite.branding.svg" Type="GhostableInLibrary" />
13  </Module>
14 </Elements>

```

Example 3 – You’ve added a custom masterpage for Addins OnPrem:

- Select file > Include in project
- Select file > Properties > Build Action: Embedded Resource



- **Elements.xml** should look like:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3   <Module Name="OnPrem" Path="OnPrem" Url="_catalogs/masterpage" RootWebOnly="true">
4     <File Path="beezy.master" Url="beezy.master" Type="GhostableInLibrary" />
5   </Module>
6 </Elements>
```

Once all new files are included, embedded and added to the Elements.xml you can proceed to build the Visual Studio solution (release or debug mode) and it will generate the branding DLL.

WSP (Full trust code)

Once the branding is built, you just have to generate the WSP package.

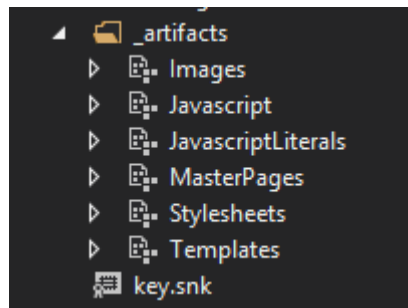
From the Visual Studio solution you just have to right-click on the SharePoint solution and select Publish.

Developing a branding package

Project structure

As mentioned in an earlier section, we only need to modify the first project of the solution in order to develop customizations in the branding package.

The most important folder is **_artifacts**:



- **Images:** folder to include new images in the branding package.
- **Javascript:** files to overwrite functionalities in Beezy components or develop new components in the branding package.
- **JavascriptLiterals:** the branding package can also overwrite the literals and translations of the product and they live in this folder.
- **MasterPages:** a branding package may have its own SharePoint master page.
- **Stylesheets:** custom design styles.
- **Templates:** the new HTML Handlebar templates (.hbs) used in the new branding package components.

Branding entry point

Inside the **JavaScript** folder there is the **beezy.branding.js** file, which is the entry point of the JavaScript code. When we create new components, they should have their own entry points and the reference to these entry points should be defined inside the **beezy.branding.js** file.

In addition, this file should also contain a reference to the **beezy.branding.scss** file.

Stylesheets

The **beezy.branding.scss** file is the entry point of the SCSS resources. This file must be used to put references to other SCSS files. By default, the file includes a reference to the **base/global** file but feel free to add new ones.

Inside this global file, we can find more references to other basic SCSS files. The most important ones are colors, typography, variables and mixins.

Webfonts

Stylesheets include a folder named **webfonts**. If we want to add our own web fonts, we have to add them here and change the **Elements.xml** file of the SharePoint module accordingly.

We must then go to **base/_typography.scss** and add the reference to the new webfont file using `@font-face`. As a best practice, it is recommended to store the font in a variable like the example font we provide in this file.

Example of **_typography.scss**:

```
1  /* THIS IS JUST A EXAMPLE OF FONT-FAMILY USE */
2
3  @font-face {
4      font-family: "sc_segoe-ui_semilight";
5      src: url("../webfonts/segoe-ui/west-european/semilight/latest.eot");
6      src: url("../webfonts/segoe-ui/west-european/semilight/latest.eot?#iefix") format("embedded-opentype"),
7           url("../webfonts/segoe-ui/west-european/semilight/latest.woff") format("woff"),
8           url("../webfonts/segoe-ui/west-european/semilight/latest.ttf") format("truetype");
9      font-weight: normal;
10     font-style: normal;
11     -webkit-font-smoothing: antialiased;
12 }
13
14
15 $font-family-segoe: "sc_segoe-ui_semilight", "Helvetica Neue", Helvetica, Arial, sans-serif;
16
```

Example of **Elements.xml**:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3      <Module Name="Stylesheets" Path="Stylesheets\beezy" Url="Style Library/beezy" RootWebOnly="True">
4          <File Path="webfonts\segoe-ui\west-european\semilight\latest.eot" Url="webfonts/segoe-ui/west-european/semilight/latest.eot" Type="GhostableInLibrary" />
5          <File Path="webfonts\segoe-ui\west-european\semilight\latest.ttf" Url="webfonts/segoe-ui/west-european/semilight/latest.ttf" Type="GhostableInLibrary" />
6          <File Path="webfonts\segoe-ui\west-european\semilight\latest.woff" Url="webfonts/segoe-ui/west-european/semilight/latest.woff" Type="GhostableInLibrary" />
7          <File Path="beezy.loadingstates.css" Url="beezy.loadingstates.css" Type="GhostableInLibrary" />
8      </Module>
9  </Elements>
```

Images

In order to add new images to the branding package, we must put the new image files in the **beezy** sub-folder within the **Images** folder. After including the new images, we must change the **Elements.xml** accordingly.

For example, if we want to change Beezy logo in our branding, we only have to go to **Images** folder and add a new image with the name:

logo-beezy-home.png

Note that the logo might not be reflected in the provider hosted add-in environments automatically so you might need to go through SharePoint site settings to enable it. However, it should be uploaded as the build succeeds.

Example of **Elements.xml**:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3    <Module Name="Images" Path="Images" RootWebOnly="TRUE" Url="Style Library/Images/beezy">
4      <File Path="beezy\Barcelona1.jpg" Url="Barcelona1.jpg" Type="GhostableInLibrary" />
5      <File Path="beezy\Barcelona2.jpg" Url="Barcelona2.jpg" Type="GhostableInLibrary" />
6      <File Path="beezy\Barcelona3.jpg" Url="Barcelona3.jpg" Type="GhostableInLibrary" />
7      <File Path="beezy\Barcelona4.jpg" Url="Barcelona4.jpg" Type="GhostableInLibrary" />
8      <File Path="beezy\Barcelona5.jpg" Url="Barcelona5.jpg" Type="GhostableInLibrary" />
9      <File Path="beezy\logo-beezy-home.png" Url="logo-beezy-home.png" Type="GhostableInLibrary" />
10   </Module>
11 </Elements>

```

Colors

In the file **_colors.scss** we have a list of variables whose name is associated with the corresponding color. Below that, the variables are associated to elements like buttons, link and extras. Feel free to use these variables or to add your own variables of color to use in your branding package.

Icons

To generate the site icons, we have to replace the SVG files inside the folders within the path **your_branding_project/sprites/src/img/svg**. These folders are divided into two classes:

no-conversion

The icons in this folder keep the original colors of the SVG files.

to-[color]-and-[hover-color]

The icons in this folder ignore the default color of the SVG and it is replaced by the color name that we put in [color] and generates a variant of the icon with the color that we put in [hover-color] for icon hover behavior.

Once we put the icons in the folder named with the corresponding colors, we have to modify the file **branding.json**, in which for each folder, must be an object with the name of this folder and the colors that correspond to the name of this folder.

This object will be within colors object. It is good practice to add the color name in **colors.scss** file if it doesn't exist yet.

Example of an icon creation

We will create a set of black icons that will go white on hover. We have to create a folder within **sprites/src/img/svg** with the following name: **to-myblack-and-mywhite**.

Then we have to add the object in the file **your_branding_project/sprites/branding.json**:

```

1 {
2   "paths": {
3     "result": "src/result/",
4     "srcDirIMG": "src/img/",
5     "destDirIMG": "...\\src\\Beezy.Branding\\_artifacts\\Images\\bezy",
6     "destDirSprite": "...\\src\\Beezy.Branding\\_artifacts\\StyleSheets\\scss\\utils\\",
7     "spriteDir": "...\\Images\\bezy"
8   },
9   "colors": {
10     "to-myblack-and-mywhite": {
11       "options": {
12         "colors": {
13           "base": "#000000",
14           "hover": "#FFFFFF"
15         },
16         "prepend": false,
17         "stateToken": "-"
18       },
19       "files": {
20         "<%= project.srcDirIMG %>svg-colors": ["<%= project.srcDirIMG %>svg-compressed/to-myblack-and-mywhite/*"]
21       }
22     },
23     "to-sungrey-and-linkcolor": {
24       "options": {
25         "colors": {
26           "base": "#000000",
27           "hover": "#000000"
28         },
29         "prepend": false,
30         "stateToken": "-"
31       },
32       "files": {
33         "<%= project.srcDirIMG %>svg-colors": ["<%= project.srcDirIMG %>svg-compressed/to-sungrey-and-linkcolor/*"]
34       }
35     },
36     "to-silver-and-linkcolor": {
37       "options": {
38         "colors": {
39           "base": "#000000",
40           "hover": "#000000"
41         },

```

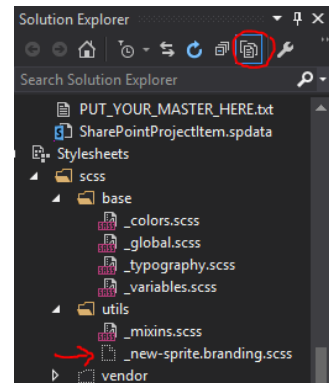
Note: Object name is underlined but the folder name uses hyphens.

The **sprite.branding.png** file is automatically created and it contains all the icons needed. The classes for each icon (taken from the SVG icon file

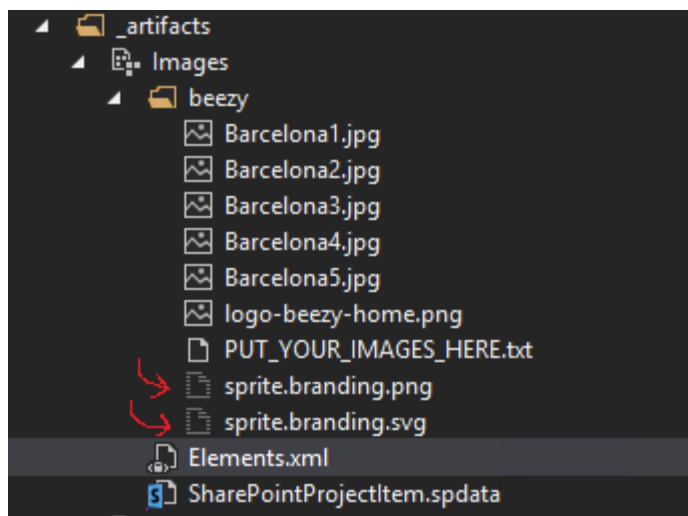
name) and its coordinates are in **_new-sprite.branding.scss** file. This file should not be manually modified as it will be regenerated automatically by the build.

If the sprites are not automatically generated, execute the file **buildsprites.bat** within the sprites folder. This will generate the SVG, PNG and SCSS files needed to use the sprites.

Remember to include this file in your branding package so that the new CSS styles are applied. If you don't see this file in your Visual Studio solution make sure that you have the option of "Showing all files" enabled as it might be hidden.



You have also to include the generated sprite files (SVG and PNG) to the project:



And also to the **Elements.xml** file:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3   <Module Name="Images" Path="Images" RootWebOnly="TRUE" Url="Style Library/Images/beezy">
4     <File Path="beezy\Barcelona1.jpg" Url="Barcelona1.jpg" Type="GhostableInLibrary" />
5     <File Path="beezy\Barcelona2.jpg" Url="Barcelona2.jpg" Type="GhostableInLibrary" />
6     <File Path="beezy\Barcelona3.jpg" Url="Barcelona3.jpg" Type="GhostableInLibrary" />
7     <File Path="beezy\Barcelona4.jpg" Url="Barcelona4.jpg" Type="GhostableInLibrary" />
8     <File Path="beezy\Barcelona5.jpg" Url="Barcelona5.jpg" Type="GhostableInLibrary" />
9     <File Path="beezy\logo-beezy-home.png" Url="logo-beezy-home.png" Type="GhostableInLibrary" />
10    <File Path="beezy\sprite.branding.png" Url="sprite.branding.png" Type="GhostableInLibrary" />
11    <File Path="beezy\sprite.branding.svg" Url="sprite.branding.svg" Type="GhostableInLibrary" />
12  </Module>
13 </Elements>

```

If you need an icon with more than two states (for instance: original – hover states plus "active" state), we have to put the icon in the folder for original and hover, and a copy with the icon with his color corresponding to active state in **no-conversion-folder** and then add "-active" to the file name. Finally, you can use the class `.[name of icon]-active` in your CSS file.

You have more information regarding **branding.json** functionalities in **ABOUT_EXAMPLE_SOURCE_CODE.txt** file inside **sprites** folder.

Variables and mixins.

In **_variables.scss** file contains the variables regarding general measures of the site, such as paddings, font heights, main banner size, etc. Note that changing these values will have no effect on the layout until you include them in your own SCSS.

For example, to change the banner height it is not enough changing the value of `$blackBannerHeight` in **_variables.scss**, you also have to include in your SCSS this value:

```
1 // -----
2 // DEFAULT CONTENT.
3 // -----
4 @import "base/global";
5
6 //Your code here. Delete existing example and add your custom imports here.
7
8
9 html:not(.ms-dialog) body:not(.ms-dialogBody) #banner-container {
10   @include prem(height, $blackBannerHeight);
11 }
```

All the values in **_variables.scss** have no units since we use the mixin `prem` to add px and rem unit to these values.

In the **_mixins.scss** file we have a group of useful mixins. Feel free to take a look to `prem` mixin to check his behavior.

You can also set this value with default CSS without using mixin or variables for example:

```

1 // -----
2 // DEFAULT CONTENT.
3 // -----
4 @import "base/global";
5
6 //Your code here. Delete existing example and add your custom imports here.
7
8 html:not(.ms-dialog) body:not(.ms-dialogBody) #banner-container {
9     height : 120px;
10 }

```

Note that we have excluded the *.ms-dialog* and *.ms-dialogBody* classes. The reason for this is because **#banner-container** is an own SharePoint id and we want to prevent our style from affecting SharePoint dialogs.

For all styles that affect Beezy elements, it is recommended to include them within a *.bz-webpart* class.

Masterpages

To modify the master in our branding, we have to download the file **beezy.master** from SharePoint, go to *Setting – Site setting - Master pages and page layouts* find **beezy.master** file. Right-click on it and select *Download a copy* from the emergent menu.

If we are developing for an onpremises add-in deployment, we should create inside **MasterPages** of the solution a subfolder called **OnPrem**. In case it is a cloud environment, the name of the folder should be **Cloud**. If the environment is WSP, the master page can be put inside of **MasterPages**. We have to verify that the **Elements.xml** file is changed automatically or change it manually.

Example of **Element.xml** (just for *WSP*):

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3   <Module Name="MasterPages" Path="MasterPages" Url="_catalogs/masterpage" RootWebOnly="true">
4     <File Path="beezy.master" Url="beezy.master" Type="GhostableInLibrary" />
5   </File>
6 </Module>
7 </Elements>

```

Example of **Elements.xml** (for Add-in OnPrem):

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3   <Module Name="MasterPages" Path="MasterPages" Url="_catalogs/masterpage" RootWebOnly="true">
4     <File Path="OnPrem\beezy.master" Url="beezy.master" Type="GhostableInLibrary" />
5   </Module>
6 </Elements>

```


Multibranding

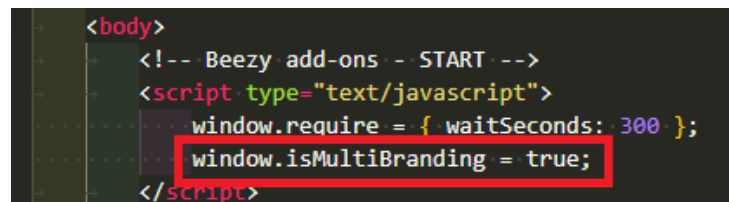
To have multibranding working there are some steps to be performed in the branding creation.

Changes on the Default's Entity branding

Changes in masterpage

All local entities will have the same masterpage, therefore it will not be possible to apply different changes according to different local entities. However, it will still be possible to make changes in the masterpage, which will apply to all local entities.

Furthermore, there is one change that must be added to the masterpage in order to enable the multibranding feature. Follow the SDK documentation instructions to download the masterpage file, and add the following code inside any of the existing `<script>` tags:



```
<body>
  <!-- Beezy add-ons - START -->
  <script type="text/javascript">
    window.require = { waitSeconds: 300 };
    window.isMultiBranding = true;
  </script>
```

Add this file to the provisioning project.

Custom logo for each local entity

There can only be one logo file across the Beezy installation. The workaround to customize it by local entity would be to upload a transparent image to the Style Library replacing the default logo in *Style Library – Images – beezy – logo-beezy-home.png*, where *logo-beezy-home.png* would be the transparent image, and then replacing that image via JavaScript branding with another image file uploaded to the Style Library.

For instance:

```
define(function() {
  const mappings = window.BeezyApp.mappings;
  return window.BeezyApp.define(
    'customLogo',
    [mappings.product.javascript.utils.image.js],
    function (ImageHelper){
      const logo = window.document.querySelector('#DeltaSiteLogo img');
      logo.src = ImageHelper.getAbsoluteURL('/Style%20Library/Images/beezy/logo-
beezy-home-alternative.png');
    }
  );
});
```

Changes on the Local Entities' branding

You must create a brand-new branding project for each local entity. You must build the branding in the usual way and when it is ready follow these steps to prepare for the multibranding.

Custom Logo

In case of a local entity, the function should be like this:

```
define(function() {
  $('link[rel="shortcut icon"]').attr('href', '../Style Library/Images/beezy/LocalEntity
/beezy-logo-ico.ico');
  const mappings = window.BeezyApp.mappings;
  return window.BeezyApp.define(
    'customLogo',
    [mappings.product.javascript.utils.image.js],
    function (ImageHelper){
      const logo = window.document.querySelector('#DeltaSiteLogo img');
      logo.src = ImageHelper.getAbsoluteURL('/Style%20Library/Images/beezy/LocalEnti
ty/logo-beezy-home.png');
    }
  );
});
```

Where the localentity on the url is the value you put on the local entity configuration. This translates to a folder in the Style Library -> Images -> beezy where you should put all the images for this branding.

Branding.json

Add the name of the local entity as in the example below

```
"paths": {  
  "result": "src/result/",  
  "srcDirIMG": "src/img/",  
  "destDirIMG": "../Ceratizit.Beezy.Branding/_artifacts/Images/beezy/LocalEntity",  
  "destDirSprite": "../Ceratizit.Beezy.Branding/_artifacts/Stylesheets/scss/utils/",  
  "spriteUrl": "../../images/beezy/LocalEntity"  
},
```

Production.config & debug.config

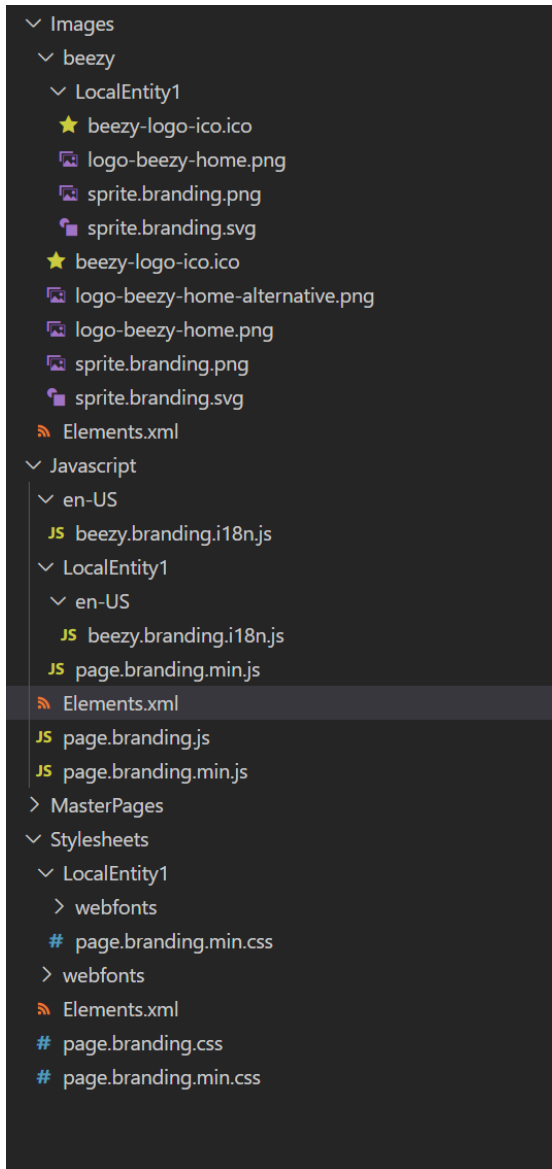
Add the name of the local entity and ../ as in the example below

```
module: {  
  loaders: [  
    {  
      test: /\. (png|gif|svg) $/,  
      loader: path.join(__debug, 'node_modules/file-loader'),  
      query: {  
        name: '../Images/beezy/LocalEntity/[name].[ext]'  
      }  
    },  
    {  
      test: /\.scss$/i,  
      loader: extractCSS.extract(['css', 'sass'])  
    }  
  ]  
}
```

Provisioning package

After you have all the LE brandings ready, you must include them in the provisioning package.

They should look like this:



You also must configure Elements.xml files in order to deploy all the files.

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Module Name="Javascript" Url="Style Library/beezy/js" RootWebOnly="true" Path="Javascript">
    <File Path="page.branding.min.js" Url="page.branding.min.js" Type="GhostableInLibrary" />
    <File Path="en_US\beezy.branding.i18n.js" Url="en-
us/beezy.branding.i18n.js" Type="GhostableInLibrary" />

    <File Path="LocalEntity1\page.branding.min.js" Url="LocalEntity1/page.branding.min.js" Type="Ghos
tableInLibrary" />
    <File Path="LocalEntity1\en_US\beezy.branding.i18n.js" Url="LocalEntity1/en-
us/beezy.branding.i18n.js" Type="GhostableInLibrary" />
  </Module>
</Elements>
```

The folder name is the value we will set in the LE configuration.

Troubleshooting

Out of memory in build process

- Check you have access to your mapped drive from the console (Administrator has not access to drives mapped from others users)
- Add the following parameters when execute node into the build.bat:
"--max_old_space_size=4096"

I changed files but they are not reflected in my Beezy environment

- Check the versioning of style library to ensure it is disabled. To do so, go to *Site contents > Style Library > Library Settings (in the ribbon) > Versioning settings > Require documents to be checked out before they can be edited? > No*.
- For environments deployed in release add "--build=release" parameter when executing build.bat

ENOENT errors in build process

- Check the permissions in your development folder.

Installation of the branding package fails

It may happen that when running ./INSTALL.BAT the npm dependencies fail to resolve. In that case, run the following command in a PowerShell window:

npm clean cache

This will clean any packages that were installed before that could be conflicting with the ones that we need for the branding.