# UPGRADING AN EXISTING 6.2 THEME TO DXP

# MIGRATING 6.2 THEMES

- In DXP, upgrading themes from 6.2 is made much easier.
- Using the Liferay Theme Generator, developers can import and upgrade themes that were originally created in the 6.2 plugins SDK.
- Let's take a look at what a developer needs to do to upgrade an existing theme.

# IMPORTING WITH THE LIFERAY THEME GENERATOR

- Developers need to import their theme first before upgrading.
- Importing includes the following steps:
  1. First, you need to navigate to the 6.2 theme directory and run the following:

     ```
     yo liferay-theme:import
     ```

  2. Next, you'll need to enter the path to the theme and hit enter.
     - The theme's modified files are all copied and migrated to a newly created `src` directory.
     - A `gulpfile.js`, `liferay-theme.json`, `package.json` file and a `node_modules` directory is added to the existing theme.
  3. Finally, you'll need to enter the path to your app server as well as the URL.
- The theme can now use gulp tasks, but needs a few upgrades to function on DXP.

# GULP UPGRADE

- With the theme set to use gulp tasks, developers can take advantage of the upgrade tasks.
  1. In the theme's root directory, you can run the following:

     ```
     gulp upgrade
     ```

     - The theme's files are placed in a `_backup` folder.
     - If you want to revert to the 6.2 theme, you can use `gulp upgrade:revert`.
  2. Hitting Enter will rename all the existing `.css` files to `.scss`.
     - All Sass files now use the `.scss` extension and Sass partials are indicated with a `_` at the beginning of the file name.
- The upgrade tasks will check all the theme's files and either upgrade or leave suggestions.
- To comply with DXP, the theme's Bootstrap code is also upgraded from version 2 to version 3.

# USING FONT AWESOME

- If Font Awesome was in use in the previous theme, developers will need to put the following variables in the `_aui_variables.scss`:

```
// Icon paths
$FontAwesomePath: "aui/lexicon/fonts/alloy-font-awesome/font";
$font-awesome-path: "aui/lexicon/fonts/alloy-font-awesome/font";
$icon-font-path: "aui/lexicon/fonts/";
```

# UPDATING THE RESPONSIVE CSS

- In DXP, the `respond-to` mixins have been replaced with explicit media queries.
- Developers will need to update their responsive CSS to the following:

1. `@include respond-to(phone)` : `@include media-query(null, $screen-xs-max)`
2. `@include respond-to(tablet)` : `@include media-query(sm, $screen-sm-max)`
3. `@include respond-to(phone, tablet)` : `@include media-query(null, $breakpoint_tablet - 1)`
4. `@include respond-to(desktop, tablet)` : `@include sm`
5. `@include respond-to(desktop)` : `@include media-query($breakpoint_tablet, null)`

# THE RESOURCES IMPORTER

- The Resources Importer has undergone serveral structural changes.
- This will impact both the configuration files and the directory structure.
- Let's look at what we need to update to make it all come together on DXP.

# UPDATING THE PLUGIN PACKAGE PROPERTIES FILE

- In the `liferay-plugin-package.properties` file, the following line should be removed:

  `required-deployment-contexts`

  - This property is no longer required, as the Resources Importer is now an OSGi module that is included in Liferay.

- The value of the `resources-importer-target-class-name` property needs to be updated to the following:

  `com.liferay.portal.kernel.model.Group`

# RESOURCES IMPORTER CONTENT XML

- In previous version of Liferay, web content was included in the Resources Importer by including `html` files.
- In DXP, all articles need to include a structure, template, and an `xml` file instead of `html`.
- Upgrading articles can easily be done by renaming the files to `[article-name].xml` and using the following pattern:

```
<?xml version="1.0"?>
<root available-locales="en_US" default-locale="en_US">
        <dynamic-element name="content" type="text_area"
        index-type="keyword" index="0">
                <dynamic-content language-id="en_US">
                        <![CDATA[
                        HTML CONTENT GOES HERE
                        ]]>
                </dynamic-content>
        </dynamic-element>
</root>
```

# STRUCTURES: FROM XML TO JSON

- In DXP, structures need to be JSON files.
- Developers can find the JSON source of any upgraded structures in *Menu→Site Administration→Content→Structures*.
- All Structure files need to be created as `[structure-name].json` and go in the `resources-importer/journal/structures/` directory.
- As mentioned above, every content article needs a structure and template.
- Let's look at the basic content structure below.

# BASIC WEB CONTENT STRUCTURE: TOP

```
{
    "availableLanguageIds": [
        "en_US"
    ],
    "defaultLanguageId": "en_US",
    "fields": [
        {
            "label": {
                "en_US": "Content"
            },
            "predefinedValue": {
                "en_US": ""
            },
            "style": {
                "en_US": ""
            },
            "tip": {
                "en_US": ""
            },
            ...
```

# BASIC WEB CONTENT STRUCTURE: BOTTOM

```
            ...
            "dataType": "html",
            "fieldNamespace": "ddm",
            "indexType": "keyword",
            "localizable": true,
            "name": "content",
            "readOnly": false,
            "repeatable": false,
            "required": false,
            "showLabel": true,
            "type": "ddm-text-html"
        }
    ]
}
```

- The structure above defines some basic attributes for the web content, sets the input field data as `html`, and identifies the web content by the ``name'': ``content''.

# CONTENT TEMPLATES

- Developers can place their existing `.ftl` templates in the `resources-importer/journal/templates` folder.
- This will also work with any `.vm` templates.
- At the very least, there must be a basic Web Content Template with the following:

  ```
  ${content.getData()}
  ```

- And with that, your theme is ready to deploy on DXP!

Notes: