ALLOYUI AND CODE UPGRADE TOOLS

LIFERAY UPGRADE TOOLS

- Whether you need to upgrade to the latest version of *AlloyUI* for Liferay 6.2 or are looking to update your code for Liferay DXP, Liferay provides the tools you need to get you started.
- In the following exercises, we'll walk through installing and running the *Liferay AUI Upgrade Tool* for updating *AlloyUI* for Liferay 6.2.
- We'll also take a look at using the *Code Upgrade Tool* to help get your code ready for Liferay DXP.

THE LIFERAY AUI UPGRADE TOOL

- When upgrading from Liferay 6.1 to 6.2, there are few *AlloyUI* changes that need to be made in the plugin code in order to successfully run existing portlets in Liferay 6.2.
  - *AlloyUI* is a framework built on top of YUI3 (JavaScript) that uses Bootstrap (HTML/CSS) to provide a simple API for building high scalable applications.
- Most of the required *AlloyUI* changes will be made in JS, CSS, and JSP files.
- In order to make upgrading your code as easy as possible, Liferay has provided the *Liferay AUI Upgrade Tool*.
- Depending on your code, the tool should be able to make about 80-100% of the required code changes.
- For more information on the tool, documentation is available at *https://github.com/liferay/liferay-aui-upgrade-tool*

EXERCISE: INSTALLING THE LIFERAY AUI UPGRADE TOOL

- You will need to have *NodeJS* installed on your machine to install the LAUT tool.

1. **Open** a Terminal/Command Prompt window.
2. **Run** the install command to install the *Liferay AUI Upgrade Tool*:

```
$ [sudo] npm install -g laut
```

```
C:\Users\Liferay>npm install -g laut
C:\Users\Liferay\AppData\Roaming\npm\laut -> C:\Users\Liferay\AppData\Roaming\npm\node_modules\laut\bin\laut.js
C:\Users\Liferay\AppData\Roaming\npm\laut-pkg -> C:\Users\Liferay\AppData\Roaming\npm\node_modules\laut\package\package.js
C:\Users\Liferay\AppData\Roaming\npm
`-- laut@1.0.6
  +-- commander@2.0.0
  +-- fs-extra@0.7.0
  | +-- jsonfile@1.0.1
  | `-- mkdirp@0.3.5
  +-- ncp@0.4.2
  +-- rimraf@2.2.5
  +-- sha1@1.1.0
  | +-- charenc@0.0.2
  | `-- crypt@0.0.2
  `-- walkdir@0.0.7
```

EXERCISE: RUNNING LAUT

- Once installed, you can run LAUT against a plugin or set of plugins.
- When run, the tool will update plugin code for the Liferay 6.2 standard.

1. **Run** the `laut` command to execute the upgrade tool:

   ```
   $ laut -f projects/liferay/liferay-plugins
   ```

   - In this example, `projects/liferay/liferay-plugins` is the directory containing the portlets which are to be updated for Liferay 6.2.

- As a developer, you will have to review changes to accept or reject.
- Even if you reject any changes, the information provided will still be useful as a hint that there may be issues in your code and that you may have to manually apply changes.

HELPFUL LAUT COMMANDS

- The *Liferay AUI Upgrade Tool* supports various commands and options:

-h, --help	Outputs usage information
-f, --file [file name]	The file(s) to process
-e, --ext [file extentions]	The file extentions that should be processed
-V, --version	Outputs the version number

```
C:\Users\Liferay>laut -h

  Usage: laut.js [options]

  Options:

    -h, --help                    output usage information
    -f, --file [file name]        The file(s) to process.
    -e, --ext [file extensions]   The file extensions which should be processed. Defaults to "js, jsp, jspf, css, tpl".
    -V, --version                 output the version number
```

WHAT'S BEING CHANGED?

- What are the changes the *Liferay AUI Updgrade Tool* is making in the code?
  - Removes the `aui-` prefix from CSS classes in CSS, JS, and JSP pages
  - Adds `-deprecated` suffix to all deprecated modules in *AlloyUI* 2.0
    - The user can configure these. They are described in JSON format in the `assets/deprecated-modules.json` file.
  - Renames CSS classes that require renaming
    - The user can configure these. They are described in JSON format in the `assets/css-classes.json` file.
  - Replaces the `inputCssClass` attribute in `<aui:input>` as the attribute is no longer used

ADDITIONAL CODE CHANGES

- Additional changes made by the *Liferay AUI Updgrade Tool*:
  - Replaces `.selector-button input` where delegate events are attached (or single listeners via `.on`) with `.selector-button`
  - Changes `handler: function(...` to `on : { click: function(... }`
    - This is usually used when adding children to AUI Toolbar.
  - Replaces all occurences of new `A.Dialog` with `Liferay.Util.Window.getWindow`
  - Adds `<portlet:namespace />` to `name` attribute of `input` elements if they are not already namespaced

THE CODE UPGRADE TOOL

- Liferay also provides the *Code Upgrade Tool* as a feature in *Liferay IDE* (included in versions 3.1 and later).
- The *Code Upgrade Tool*:
  - Identifies code affected by the DXP API changes
  - Describes each API change related to the code
  - Suggests how to adapt the code
  - Provides options, in some cases, to adapt code automatically
- Even if you use tools other than *Liferay IDE*, you should upgrade Plugins SDK plugins using the *Code Upgrade Tool* in *Liferay IDE* first. You can use your favorite tools later.

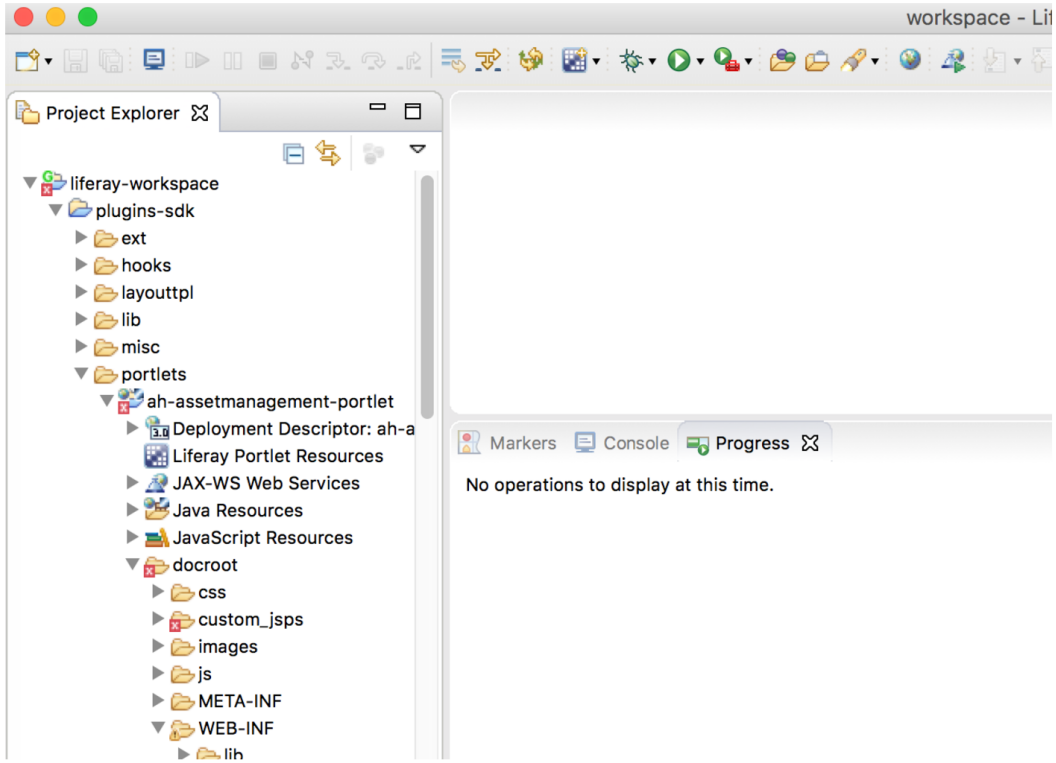
EXERCISE: UPGRADING EXISTING CODE FOR DXP

- Once you have updated code with the *Liferay AUI Upgrade Tool* for Liferay 6.2, you can use the *Code Upgrade Tool* to adapt your existing code for Liferay DXP's new API standards.

1. **Copy** a newly upgraded plugin from your Liferay 6.1 Plugins SDK to your Liferay workspace in the *plugins-sdk* directory.
    - This directory houses the legacy plugins within your Liferay workspace.
    - You can specify the location in your `gradle.properties` file at the root of your Liferay workspace by adding the following property:
        - `liferay.workspace.plugins.sdk.dir=plugins-sdk`
2. **Start** *Liferay IDE*.

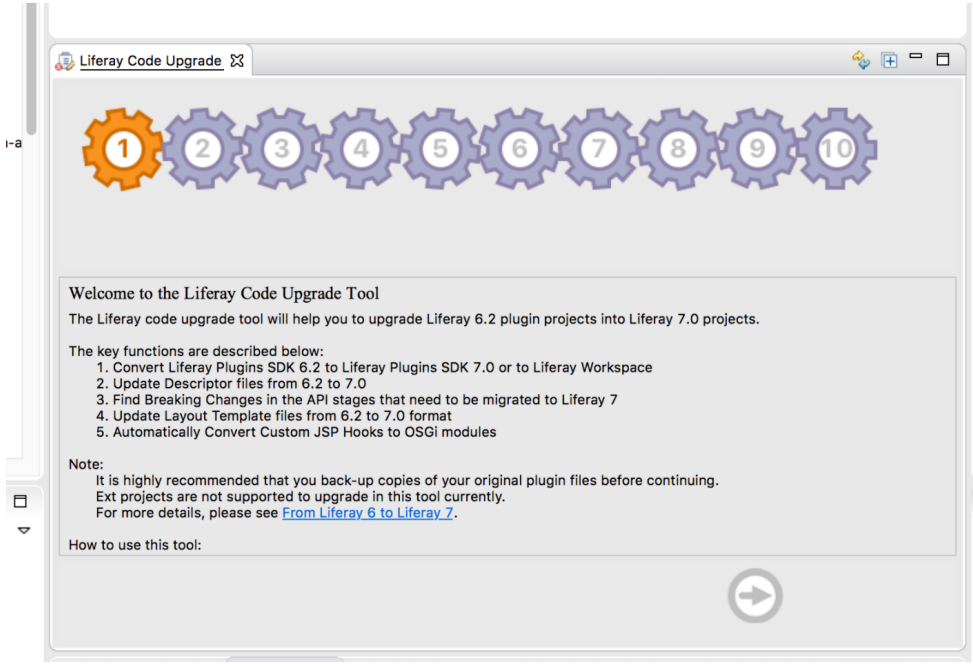
EXERCISE: CREATING A NEW WORKSPACE

1. **Create** a new workspace that points to your existing Liferay workspace.
   - Alternatively, you can create an empty workspace and add the Liferay workspace to your project.
   - You should see something like this once the workspace is created:



EXERCISE: THE LIFERAY CODE UPGRADE TOOL

1. Go to *Project→Liferay Code Upgrade Tool* in the menu.
   - The tool will open in a new tab within the IDE.
   - You can either expand the tab or drag it to a center window for easier use.

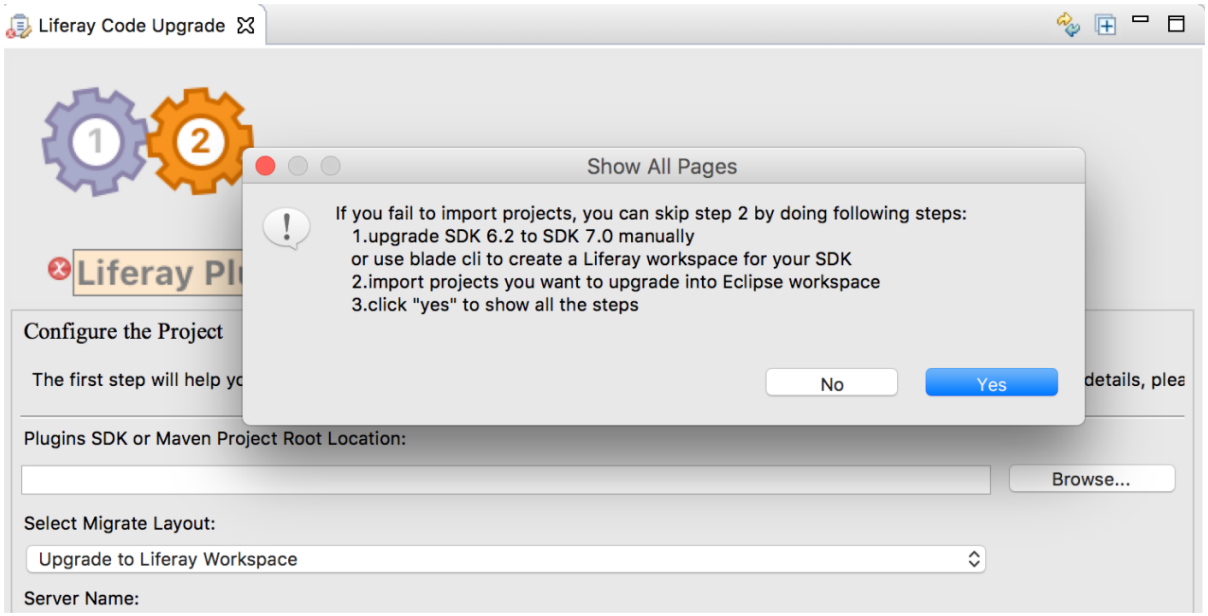


2. Click on the right arrow at the bottom of the screen to begin.

EXERCISE: THE IMPORTED WORKSPACE

- Because you have already imported your existing Liferay workspace into the *Liferay IDE* workspace, you can bypass the import screen and error message.

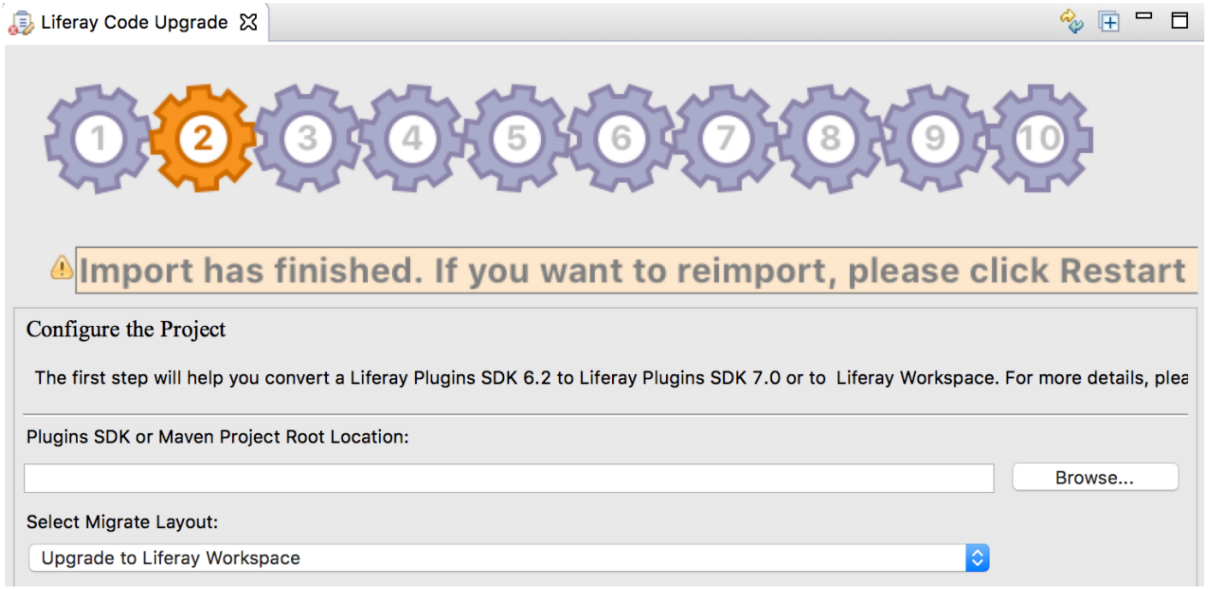
1. **Click** on the plus icon at the upper right corner of the tab.
2. **Click** *Yes* on the pop-up.



EXERCISE: CONFIRMING THE IMPORTED WORKSPACE

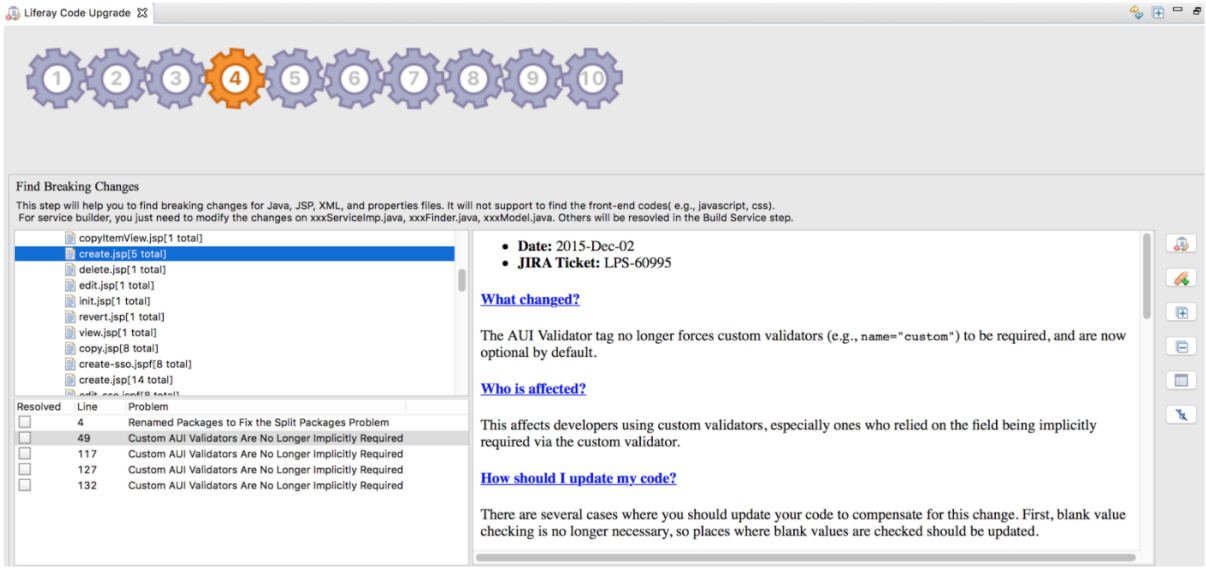
- To verify that the previous step was bypassed, you can click back to step 2 by clicking the icon labeled 2.

1. **Click** on the gear icon labeled 2 to confirm the import.
   - You should see that the import has finished.



THE NEXT STEPS

- You can now continue going through the *Code Upgrade Tool* steps.
- For more information on the *Code Upgrade Tool* and an explanation of each step in the process, documentation is available at *https://dev.liferay.com/develop/tutorials/-/knowledge_base/7-0/adapting-to-liferay-7s-api-with-the-code-upgrade-tool*



ADDITIONAL MANUAL CHANGES

- Once you have run through the steps for *Code Upgrade Tool*, you may still need to take your upgraded DXP plugins and make additional manual changes or fixes.
- You may also need to deploy as a WAB or convert to an OSGi module.
- When using Liferay's upgrade tools, you'll be well on your way to upgrading to the latest version of the Liferay Platform.

Notes: