How to Upgrade to Liferay Digital Experience 7.3 from Liferay 6.x

Table of Contents

Introduction

As customer demand for always-on and everywhere digital experiences has risen, so has the need for companies to equip themselves with the right technologies to deliver great digital experiences and remain agile for future digital innovations. Liferay Digital Experience Platform (DXP) enables digital businesses to manage and deliver customer experiences that are consistent and connected across digital touchpoints, including mobile, desktop, kiosk, smart devices, and more.

An upgrade to Liferay DXP is an investment in addressing immediate needs in digital experience while laying the foundation to serve an increasingly connected digital audience. The upgrade places your business in the best position to take advantage of Liferay's latest developments for digital businesses including in headless APIs, asset auto-tagging, content recommendations, advanced segmentation and personalization capabilities, and more.

This whitepaper aims to set a framework for Liferay's recommended upgrade path for your organization. A major technology upgrade is an endeavor requiring a deep analysis of your business requirements, careful planning, testing, and execution in order to be successful. Before you start on your planning and execution, Liferay's Global Services team, our group of professional consultants with extensive experience in upgrading customers to the latest Liferay platform, can help you in a critical analysis of your needs through the Liferay Upgrade Analysis Program. Pairing this comprehensive analysis with an awareness of the key considerations needed in an upgrade to Liferay DXP, you can make the most informed decisions for your company to start benefiting from the Liferay platform.

Upgrade General Timeline

Important Considerations:

Upgrades require data modifications, code modifications, and infrastructure changes. The process inherently involves a great deal of risk. Careful thought should go into *what's wanted vs. what's actually needed*.

Sufficient time should be given to do performance testing to dig up any code related conflicts or defects (this includes potential defects or bugs in Liferay DXP). This should happen before any environment tuning is conducted.

Extra time should be taken into consideration of reviewing all the customization, hooks, themes, tables, plugins, etc. The more complex the setup, the more time is needed to plan for any possible issues. In Liferay DXP, all the portal properties should be reviewed to determine the actual need/want of the functionality. Customers should talk to Liferay Support to clarify any uncertainties about what a portal property does.

Answering the following questions now can help you set a proper timeline and set expectations to better manage the risks.

1. **What version am I on?** If you are a few versions away from the current Liferay version, you should remember that your data will need to go through the full upgrade path of previous versions before you will eventually be upgraded to Liferay DXP. This means that if you are on Liferay Portal 6.1, your data will first be upgraded to 6.2 before it is upgraded to Liferay DXP. We recommend using the Upgrade tool to do so, as it can handle the complexities involved. The only use case in which you should upgrade manually is if you are upgrading a Portal version that is older than 6.1.
2. **How much data do I have?** If your project has a lot of data, it is essential to have a properly indexed database. Also set aside more time if you have a larger database.
3. **How much of my existing web content, templates and structures will I need to upgrade?** Web content, templates and structures are all upgradeable into 7.x. However, if you are on 6.1, you may have issues with the requirement to have unique elements within a structure. More information on the issue can be found here.
4. **How many portlets will I need to upgrade?** Not all portlets will need to be upgraded for Liferay DXP. A proper analysis will give you a better estimate on how much time you will need and the number of engineers to devote for this process.
5. **Am I overriding many JSPs?** JSPs have changed a lot since versions 6.1 and 6.2. We recommend moving away from overriding JSPs completely, if possible. A number of JSPs have extensions you can plug into without resorting to this.
6. **Do I have an EXT to upgrade?** The good news is that with Liferay DXP, we've created many more extension points, so the use of EXTs can be reduced. Evaluate if customizations can be done without using an EXT.
7. **Am I planning to convert to OSGi bundles?** It will take more time, but the investment may well be worth it. Not everything needs to be converted to an OSGi bundle.

General Timelines:

Architecture Review: 1–2 weeks

The internal architecture review for a project will vary but generally speaking, plan for 1–2 weeks. Developers should be aware of the new modular architecture and what that means but sometimes they do not see its implications at the time of configuring their dependencies.

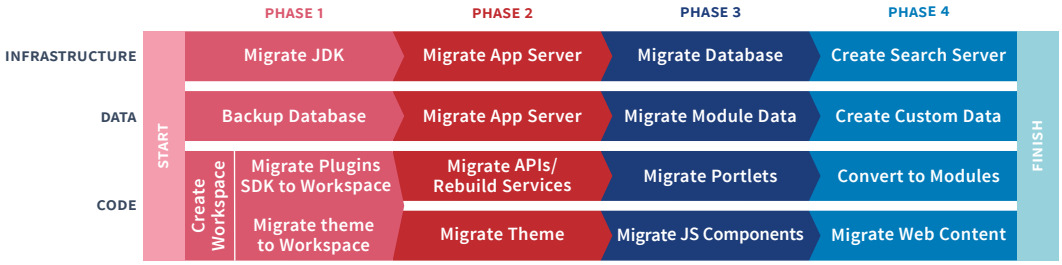
Scalability Review: 1–2 weeks

After the Architectural Review is complete, take 1–2 weeks to determine if the current code design accounts for the necessary scalability for your use case. This should be done with your code review.

Liferay Upgrade: 1–4 months

This document will take you through the Liferay Upgrade. Here you will perform the necessary changes to upgrade your data, customizations and environments to Liferay DXP 7.3. The architecture and scalability review should ensure that you make the best use of your upgrade time.

Sample Liferay DXP Upgrade Path



Testing: 1–2 weeks

This is the period of time where code should be fixed in order to get to the point to do performance tuning and load testing. Please take a look at the suggested starting points for minimal architecture and hardware platforms/requirements, as outlined in our Liferay DXP Performance Whitepaper.

Performance Tuning: 1–2 weeks

This starts once any code-related bottlenecks are resolved and configuration files need to be changed in order to find out the best values for performance. Attempting to tune performance prior to code review can lead to bottlenecks in the upgrade. During this time, we recommend going through the Liferay DXP Deployment Checklist. In addition, all other systems (database, Apache HTTPd, application server, Liferay DXP itself) will need to undergo a thorough review via your internal team specialists. The average performance tuning takes 2–4 weeks with an experienced QA team or DBA in particular. The timing can vary on your team's expertise.

Infrastructure Changes

Compatibility Matrix

Now that you're ready to begin your upgrade, first start with making sure your environment is up to date with the Liferay DXP 7.3 Compatibility Matrix. Many environments have gone end-of-life since 6.2 was released. It is essential that your system is running in a supported environment so you are able to receive proper support.

Search

The biggest change you will notice is that Liferay now requires the search engine to run as a separate JVM, either on the same or on a dedicated physical/virtual server.

Because search is a vital part of any successful deployment, we have greatly improved the search functionality in Liferay DXP. The performance and scalability profile of a search engine is drastically different from that of a portal actively serving web impressions. In the past, Liferay Portal either supported an embedded Lucene search engine or a remotely deployed Solr search engine. With Liferay DXP, we will continue to offer Solr as a supported remote search engine through DXP 7.3. However, we recommend you use Elasticsearch since Solr will not be available in future versions of Liferay. This allows the same engine to be used in embedded mode during development and in a standalone mode for production.

If you were previously using Solr, you will only need to ensure you are using the newest Solr module and upgrade to 7.x. Please refer to our search documentation to properly configure your search server. Also note that you can run your search server on the same server, but separate JVM, if you have limited resources, though this is not recommended.

JDK

Another change to the compatibility matrix is that we've moved to JDK11. All app servers are already JDK8 compatible on the compatibility matrix. Please ensure your app server is properly configured.

Deployment Plan

With the introduction of the OSGi container in Liferay DXP, your deployment plan will need to change as there are now a few different ways to deploy your plugins.

- **OSGi bundles/modules** are a new type of plugins in Liferay DXP. They are just simple Java JARs with OSGi metadata. OSGi bundles can only be deployed into the OSGi container. Bundles cannot access services deployed as WARs besides Liferay's core services. This is the recommended approach for most new development and will be the approach Liferay takes for all new development.
- **WARs** are traditional Liferay Plugin web applications (e.g., *-theme.war, *-portlet.war, *-web.war) that you have grown to love. WARs do not benefit from the explicit dependency and lifecycle management models provided by OSGi but, luckily in Liferay DXP, all WARs (except an EXT) are converted into a WAB. When you deploy a WAR file into Liferay, it automatically gets converted into a WAB.
- **WABs** are web archive bundles. This will give you all the benefits of a bundle without doing the conversion. This is the recommended approach for deploying JavaEE applications (e.g. SpringMVC or JSF), legacy applications built for older versions of Liferay, and themes.

All Liferay plugins (except an EXT) are now deployed to the OSGi container. They will either be an OSGI bundle or a WAB.

CAUTION: You should never deploy Liferay artifacts (WARs, modules, WABs) directly using your application server's deployment tools.

When deploying OSGi bundles in an application server's managed cluster (e.g., JBoss domain mode, WebLogic w/ Node Manager, WebSphere with Deployment Manager), you will need to rely upon Liferay's Cluster Deployment Helper.

This tool will take specified deployment artifacts and bundle them into a specialized WAR. When the application server deploys and starts the WAR, the startup mechanisms will place the OSGi modules into Liferay Digital Enterprise's OSGi deployment folder on the application server.

Database Upgrade

Before You Start

Now that your infrastructure has been upgraded to supported versions, we can focus our attention on the data. The first thing we should do is ensure a proper backup is in place for us in case of failures.

Next, ensure you are running permission algorithm 6 if you are coming from 6.1. If you are not on permission algorithm 6, you must migrate to that permission algorithm. Information on how to migrate permission algorithms can be found here.

Also, remember all your database indexes have been applied correctly. A missing index can cause an upgrade to really slow down. If you are running into a slow upgrade, later on, you may want to come back and add additional temporary indexes to help speed them up.

If you have staging enabled, we recommend that you publish all content before performing the upgrade.

Many of the 6.2 portal properties have been migrated to OSGi configuration. Please use the upgradeProps Blade Cli command to see which ones are affected. You can also check the traces printed by the process *VerifyProperties* during the upgrade to get more information.

Liferay DXP 7.3 already disables the search indexation so there is no need to disable it manually.

Database Upgrade Tool

In Liferay DXP, the database upgrades have been moved to a standalone tool.

To run the tool:

```
db_upgrade.sh or db_upgrade.sh
```

The upgrade requires three files to be configured before it can run: `app-server.properties`, `portal-upgrade-database.properties`, `portal-upgrade-ext.properties`. When you run the database upgrade tool, it will ask a series of questions about the installation file paths and database connection properties and create these files before starting the upgrade of the database.

```
D:\liferay\demo\liferay-7.1.10-fs\tools\portal-tools-db-upgrade-client>db_upgrade.bat
[ jboss jonas resin tcserver tomcat weblogic websphere wildfly ]
Please enter your application server (tomcat):
tomcat
Please enter your application server directory (D:\liferay\demo\liferay-7.1.10-fs\tomcat-9.0.6):

Please enter your extra library directories in application server directory (/bin):
/bin
Please enter your global library directory in application server directory (/lib):
/lib
Please enter your portal directory in application server directory (/webapps/ROOT):
/webapps/ROOT
[ db2 mariadb mysql oracle postgresql sqlserver sybase ]
Please enter your database (mysql):
mysql
Please enter your database JDBC driver class name (com.mysql.jdbc.Driver):
com.mysql.jdbc.Driver
Please enter your database JDBC driver protocol (jdbc:mysql://):
jdbc:mysql://
Please enter your database host (localhost):
localhost
Please enter your database port (none):

Please enter your database name (lportal):
/fsv2upgrade
Please enter your database username:
lruser
Please enter your database password:
*******
Please enter your Liferay home (D:\liferay\demo\liferay-7.1.10-fs):
```

The data upgrade is now broken up into two parts. The core upgrade is similar to what you've seen in the past. The next part will upgrade the OSGi modules. By default, the database upgrade tool is configured to upgrade both automatically.

Upgrade on startup

For Liferay DXP 7.3, the auto upgrade behavior can be controlled by a new portal property which covers both the Core and module upgrades:

```
Upgrade.database.auto.run
```

The default value for this property is *false* so upgrades will not run automatically on startup to provide users more control over database changes.

This property can be especially useful for container environments where an upgrade to the database is needed on startup. For this use-case, just set the property to *true*. Be sure to treat this startup as you would any regular upgrade, performing needed measures like making a backup, as well as any other pre-upgrade tasks.

This property also affects the installation of fix packs. Please read the following article to get more information about it.

Troubleshooting

If your upgrade ran successfully, you can skip this section. If you run into issues, here are some tips to help you:

- If you are trying to upgrade and running into issues, remember the tips we presented before you began. **Most upgrade issues are due to corrupted data**. The only way to fix issues of this kind is to fix or remove the corrupt data. Fortunately, in Liferay DXP 7.1 and onwards, you can resume the upgrade from where you left off.
- A common, yet easily addressed, issue with the upgrade is **properties settings that have typos**. If the upgrade is not connecting to the database or not finding the correct files in the Liferay installation, double check the properties files.

One of the advances for Liferay DXP is the separation between a logical "core" and a series of "modules." This allows us to take smaller incremental upgrades to help triage and resolve data related issues.

For more advanced techniques, you can try to use the debugger. The advantage of this is that, in some cases, you can fix the corrupt data in memory and it will be saved in the database, fixing your data without a restart. It requires a high level of background knowledge about the tables though and is only recommended for the most seasoned Liferay developers. This technique also opens up a new option during module upgrades because OSGi bundles are upgraded in steps. You can use a debugger to stop at any step and take a snapshot of your database there.

Upgrading the Modules

In the case the upgrade encounters errors in modules, you can use the Gogo shell to get more information and re-launch them once the issue is resolved. You can configure the upgrade tool to open a Gogo Shell after the upgrade by adding **-s** option:

```
./db_upgrade.sh -s or

db_upgrade.bat -s
```

You can also access the Gogo shell from the Control Panel or with *telnet localhost {port}* configuring the address with the portal *properties module.framework. properties.osgi.console*

After that, you can use the available commands in the upgrade namespace. For example:

```
        upgrade:list

        upgrade:execute

        upgrade:executeAll

        upgrade:check

        upgrade:checkAll

        verify:list

    verify:execute
```

By typing `upgrade:list`, the console will show you the modules you can upgrade since all their upgrade dependencies are covered. If you do not see any modules, that is because we need to upgrade its dependencies first. You could enter the command `scr:info {upgrade_ qualified_class_name}` to check which dependencies are unsatisfied. For example: `scr:info com.liferay. journal.upgrade.JournalServiceUpgrade`

By typing `upgrade: list {module_name}`, the console will show you the steps you have to complete for upgrading your module. To understand how this works, it can be useful to see an example; if you execute that command for the bookmarks service module, you will get this:

```
        Registered upgrade processes for com.

        liferay.bookmarks.service 1.0.0

      {fromSchemaVersionString=0.0.1, toSchemaVersionString=1.0.0-step-3,

        upgradeStep=com.liferay.bookmarks.upgrade.

        v1_0_0.UpgradePortletId@497d1106}

      {fromSchemaVersionString=1.0.0-step-1, toSchemaVersionString=1.0.0,
```

```
    upgradeStep=com.liferay.bookmarks.upgrade.
    v1_0_0.UpgradePortletSettings@31e8c69b}
  {fromSchemaVersionString=1.0.0-step-2,
  toSchemaVersionString=1.0.0-step-1,
    upgradeStep=com.liferay.bookmarks.upgrade.
    v1_0_0.UpgradeLastPublishDate@294703b6}
  {fromSchemaVersionString=1.0.0-step-3,
  toSchemaVersionString=1.0.0-step-2,
upgradeStep=com.liferay.bookmarks.upgrade.v1_0_0.UpgradeClassNames@7544b6e5}
```

This means that there is an available process to upgrade bookmarks from 0.0.1 version to 1.0.0. To complete it, you would need to execute four steps and the first one is the one that starts on the initial version and finishes in the first step of the target version (the highest step number, step-3 for this example), UpgradePortletId in this case. The latest step is the one that starts in the latest step of the target version (the lowest step number, step-1) and finishes in the target version (1.0.0), UpgradePortletSettings in this case.

By typing `upgrade:execute {module_name}`, you will upgrade a module. It is important to take into account that, if there is an error during the process, you will be able to restart the process from the latest executed step successfully instead of executing the whole process again. You could check the status of your upgrade by executing upgrade:list {module_name}.

By typing `` `upgrade:check `` at the Gogo shell, it will show you the modules that have not reached the final version. Thus you will have a way to identify the modules whose upgrades have failed at the end of the process.

To understand how this command works, consider this example: Picture that the upgrade for module com.liferay.dynamic.data.mapping.service fails in the step 1.0.0-step-2. If you execute the command upgrade:check at this moment you will get:

```
Would upgrade com.liferay.dynamic.data.mapping.service from
1.0.0-step-2 to 1.0.0 and its dependent modules
```

That means that you will need to fix the issue and execute the upgrade for that module again. Notice that dependent modules for com.liferay.dynamic.data.mapping.service need to be upgraded once the first one is upgraded properly. Also, you can execute the verify process from the command line using `verify:list`. This checks all available verify processes. Execute `verify:execute {verify_qualified_name}` to run it.

Upgrading your Code

Finally, we are covering how to upgrade your code to Liferay DXP. If you have multiple members of your team, this part can be started in tandem with the data upgrade. In order to upgrade your code, we first need to familiarize you with the Liferay Project SDK and the tools that are contained in it.

Liferay Project SDK

As part of DXP, Liferay invested heavily in improving and modernizing its developer tooling. The ANT-based Liferay Plugins SDK has been deprecated in favor of the new Liferay Project SDK, a modern approach including Gradle or Maven build toolchains that provide a more complete end to end experience.

Liferay Project SDK contains:

- Liferay Workspace (Gradle or Maven based project scaffolding for CI and DevOps).
- Brand new Gradle and Maven plugins that covers all aspects of a Liferay project's build requirements, from backend services, to portlets, to modern JS front end and themes.
- Blade CLI (new command line tool for fast developer workflows).
- Liferay Developer Studio (traditional fully integrated development environment based on Eclipse).

The Liferay Project SDK has an easy installer that will install all the required tools for Liferay DXP development. It will also give the option to set up your first Liferay Workspace. Please download the appropriate installer for your OS (Windows, Linux, OSX).

Liferay Workspace

One of your first tasks in upgrading your code is migrating it to our new project structure based on Liferay Workspace. It should be very familiar to our original Plugins SDK. It supports both Gradle and Maven and provides backwards compatibility for your Plugins SDK based projects. It will also leverage all the new theme tooling we've built in Liferay DXP and integrates with Liferay Developer Studio.

Your new project will look like this:

```
.
├── configs
│   ├── common
│   ├── dev
│   ├── local
│   ├── prod
│   └── uat
├── modules
│   └── contains new OSGi modules
├── themes
│   └── contains new node.js based frontend themes
├── wars
│   └── contains traditional portlet and theme
WARs (like those built with PluginsSDK)
├── build.gradle
├── gradle-local.properties
├── gradle.properties
└── settings.gradle
```

If you are coming from a project that leveraged the Plugins SDK, you will be able to move your Plugins SDK as one of the folders within Liferay Workspace. Keep in mind, the Plugins SDK only supports creating WARs. You will not be able to create OSGi bundles from the Plugins SDK.

Blade CLI

Alongside Liferay Workspace, we have a new command line tool called blade CLI. This tool allows you to create applications, extensions, etc. as you could in a Plugins SDK, but it also provides additional functionality. It can start your Liferay server or automatically deploy your project as you make changes. It is also the way you create a new Liferay Workspace. If you are coming from an existing Plugins SDK project structure, we have a way to automatically migrate you to the new Liferay workspace project structure as shown below.

To create a new Liferay Workspace:

```
$ blade init workspace-name
$ cd workspace-name
```

To upgrade an existing Plugins SDK to Liferay Workspace:

```
$ cd plugins-sdk
$ blade init -u
```

To create a new module:

```
$ blade create -t mvc-portlet module-name
```

Please consult our documentation to see additional commands provided by Blade.

Breaking Changes

Now that we have your code upgraded to our new project structure, perhaps the most difficult parts of upgrading your codebase is actually knowing what's changed since the last Liferay release and then making the appropriate changes to your code to support it. For Liferay DXP, we have a section on breaking changes in our documentation. It presents a chronological list of changes that break existing functionality, APIs or contracts with third-party Liferay developers or users. Some of the types of changes documented in the file include:

- Functionality that is removed or replaced.
- API incompatibilities: Changes to public Java or JavaScript APIs.
- Changes to context variables available to templates.
- Changes in CSS classes available to Liferay themes and portlets.
- Configuration changes: Changes in configuration files, like portal.properties, system.properties, etc.
- Execution requirements: Java version, J2EE Version, browser versions, etc.
- Deprecations or end of support: For example, warning that a certain feature or API will be dropped in an upcoming version.
- Recommendations: For example, recommending using a newly introduced API that replaces an old API, in spite of the old API being kept in Liferay for backward compatibility.

It is important that you familiarize yourself with the full list to understand what changes you may have to make with your codebase. We recommend reading the list of breaking changes to get a general feel of how Liferay DXP will continue to evolve in future versions. To view the current breaking changes for Liferay DXP, visit the list here.

Liferay Developer Studio

Liferay Developer Studio is the all-in-one integrated development environment for Liferay that some of you may already be using on your existing project. Our Liferay Project SDK can optionally install Liferay Developer Studio. Liferay Developer Studio includes a brand new Code Upgrade Tool to help upgrade your 6.x Plugins SDK (or maven based) project to Liferay workspace and Liferay DXP. Even if your team prefers another IDE or developer environment, we recommend you still use Liferay Developer Studio at least to use the Code Upgrade Tool during the upgrade process.

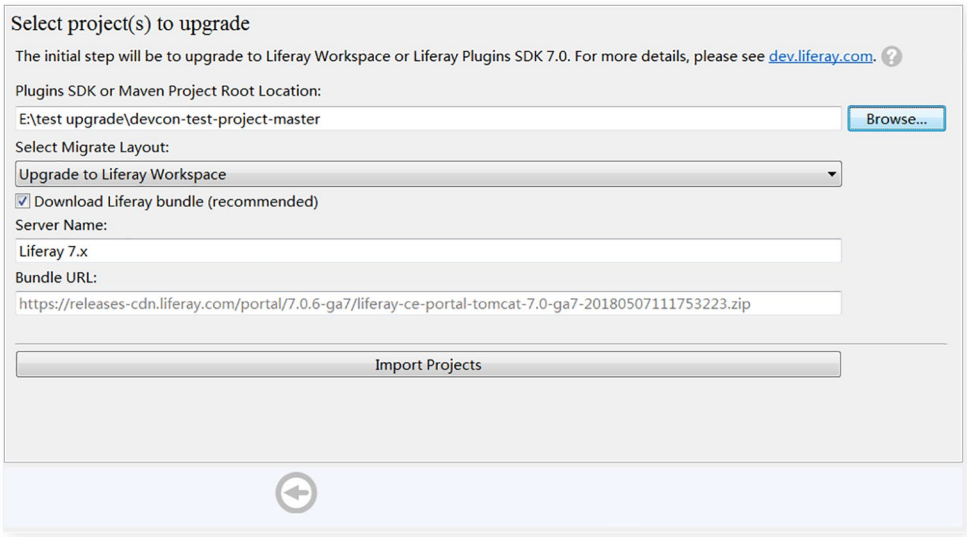
Code Upgrade Tool

The Code Upgrade Tool provides the following benefits:

- Identifies code affected by the 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

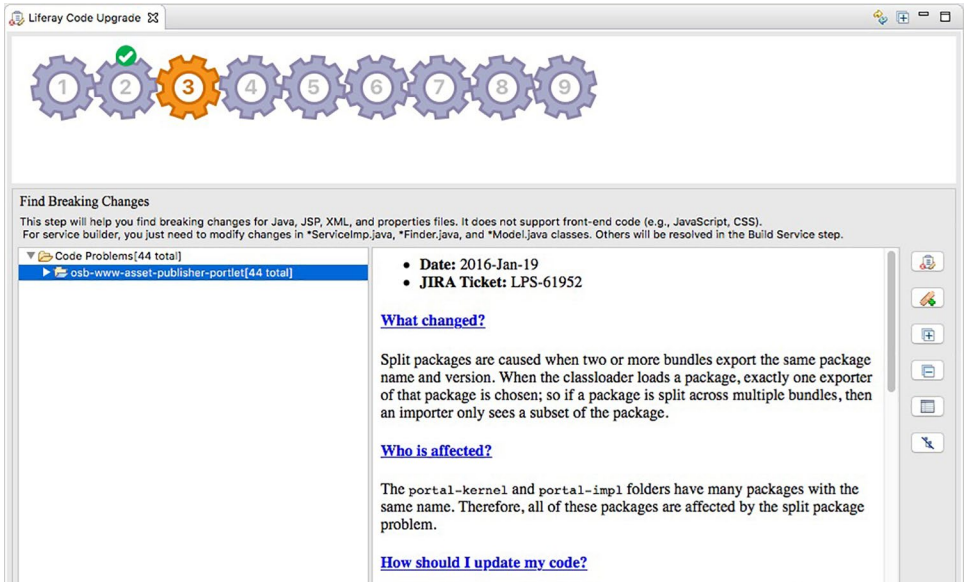
Upgrading from Plugins SDK to Liferay Workspace

All you need to do is tell the Code Upgrade Tool where your existing 6.x Plugins SDK project structure is located and it can automatically upgrade to Liferay Workspace.



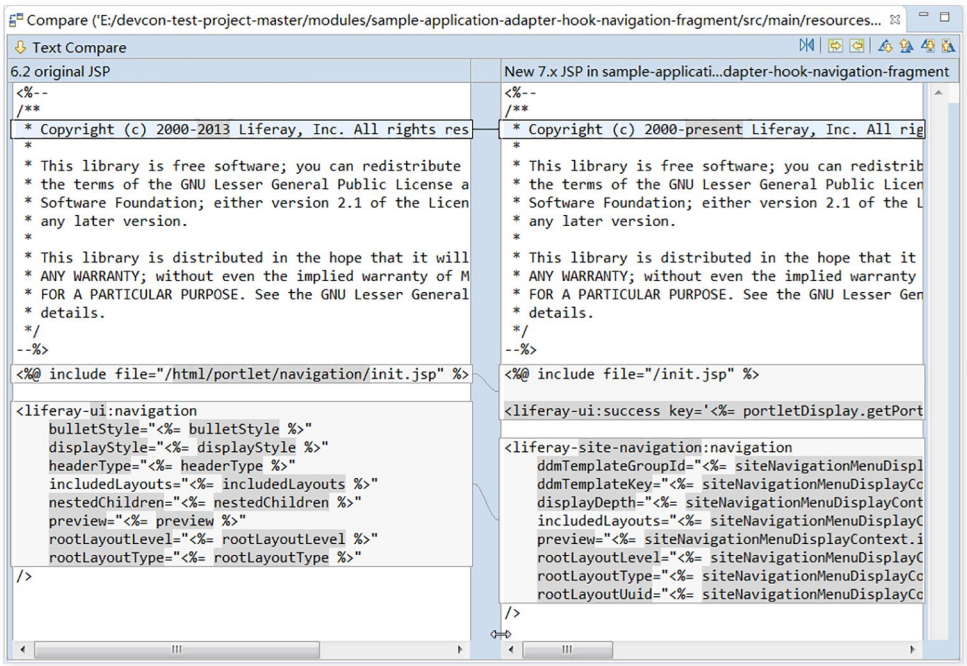
Finding Breaking Changes

Once you have given the Code Upgrade Tool your code, it will analyze it against the list of known Breaking Changes and gives you an easy way to identify, learn about breaking change and most importantly action items to take to update your code. In many cases, the tool can do this automatically for you.



Upgrading Customizations to New Modular Structure

Sometimes there is no way to upgrade your existing code in place and require a new, more modular structure, such as with JSP hooks. The Code Upgrade Tool can even help you in this very complex scenario.



For more information on how to use the Code Upgrade Tool and all of its capabilities see this detailed tutorial.

Code Upgrade Scenarios

Migrating a 6.2 WAR to a Liferay DXP-supported WAR

The first step you must take is updating your 6.2 portlet to a Liferay DXP portlet. Even if you plan on converting your portlet to OSGi modules, we recommend that you update your legacy WAR to be supported by Liferay DXP first. If you jump from a legacy 6.2 WAR to DXP modules, it will be much more difficult to debug and figure out which issues are related to API changes and which are related to the migration process. The Upgrade Assistant mentioned above to find the breaking changes in your portlet and update them accordingly. Make sure to also update any Liferay dependencies you've specified to Liferay DXP

(e.g., `ivy.xml, liferay-plugin-package.properties`, etc.). When you've completed the upgrade process and have a Liferay DXP-supported WAR, you'll need to make a decision on whether you should convert your portlet to an OSGi module. We've outlined scenarios below that will help you make your decision.

Converting a Portlet to an OSGi Module

Now that we've created our workspace and updated your portlet to a Liferay DXP supported WAR, we can consider if you want to convert your portlet to an OSGi module.

YOU SHOULD CONVERT WHEN:

- You have a very large application with many lines of code. For example, if there are many developers that are collaborating on an application concurrently and making changes frequently, separating the code into modules will increase productivity and provide the agility to release more frequently.
- Your plugin has reusable parts that you'd like to consume from elsewhere. For instance, suppose you have business logic that you're reusing in multiple different projects. Instead of copying that code into several different WARs and deploying those WARs to different customers, you can convert your application to modules and consume the services provided by those modules from other modules.

YOU SHOULD NOT CONVERT WHEN:

- You have a portlet that's JSR-168/286 compatible and you still want to be able to deploy it into another portlet container. If you want to retain that compatibility, it is recommended to stay with the traditional WAR model.
- You're using a complex legacy web framework that is heavily tied to the Java EE programming model, and the amount of work necessary to make that work with OSGi is more than you feel is necessary or warranted.
- Your plugin interacts with other JEE app server features, for instance EJBs, message driven beans, etc. Module-based applications are not as portable when they directly interact with the app server.
- Your legacy application's original intent is to have a limited lifetime. If you decide to convert your portlet to an OSGi module, we'll walk you through it here:

For a portlet: `blade create -t mvc-portlet [APPLICATION_NAME]`

If you need Service Builder: `blade create -t servicebuilder -p [ROOT_PACKAGE] [APPLICATION_NAME]`

The first thing you will notice is that projects now use the standard maven project structure.

```
.
├── bnd.bnd
├── build.gradle
└── src
    └── main
        ├── java
        │   └── com
        │       └── liferay
        │           └── samples
        │               └── servicebuilder
        │                   └── web
        │                       └── JSPPortlet.java
        └── resources
            ├── META-INF
            │   └── resources
            │       ├── css
            │       │   ├── _partial.scss
            │       │   └── main.scss
            │       ├── edit_foo.jsp
            │       ├── foo_action.jsp
            │       ├── icon.png
            │       ├── init.jsp
            │       └── view.jsp
            └── content
            └── Language.properties
```

In your workspace, the `bnd.bnd` file is very important, as it will automatically apply the liferay-gradle-plugin to your Gradle project. The liferay-gradle-plugin will apply the Gradle Java plugin along with other Liferay plugins that are very

useful such as css-builder, source-formatter, and lang-builder. You will notice that you do not need a `portlet.xml/liferay-portlet.xml` file. The contents of that file should be moved into the portlet Java class.

```
@Component(
    immediate = true,
    property = {
        “com.liferay.portlet.display-category=category.sample”,
        “com.liferay.portlet.icon=/icon.png”,
“javax.portlet.name=1”,
        “javax.portlet.display-name=Tasks Portlet”,
        “javax.portlet.security-role-
ref=administrator,guest,power-user”,
        “javax.portlet.init-param.clear-request-parameters=true”,
        “javax.portlet.init-param.view-template=/view.jsp”,
        “javax.portlet.expiration-cache=0”,
        “javax.portlet.supports.mime-type=text/html”,
        “javax.portlet.resource-bundle=content.Language”,
        “javax.portlet.info.title=Tasks Portlet”,
        “javax.portlet.info.short-title=Tasks”,
        “javax.portlet.info.keywords=Tasks”,
    },
    service = Portlet.class
)
public class TasksPortlet extends MVCPortlet {
```

If you’ve created a service builder template, you will notice two different plugins generated for you.

`plugin-name-api` - This is where the interfaces of your services will live.

`plugin-name-service` - This is where the implementations of your services will live.

The packaging will be similar to the portlet above.

Upgrading Themes

In Liferay DXP, we've created a new set of theme tools that front-end developers should be much more familiar with. They are built using Node.js, yo and gulp. If you have an existing theme in your Plugins SDK, you can migrate them to your new Liferay Workspace. The new theme tools also help facilitate the theme upgrade process. Whether upgrading from 6.x to 7.0/7.1 or 7.0 to 7.1, you would continue to utilize

```
gulp upgrade
```

However, there are some differences when upgrading from 6.1 to 7.0 versus 6.2 to 7.0. Please consult the appropriate documentation for the differences:

1. Upgrading to DXP 7.2
2. Upgrading from Liferay Portal 6.1 to 7.0

When upgrading from 7.0 to 7.1, many of your DXP 7 themes should continue working. You may wish to run the theme upgrade tools. You can find more details in the Liferay Customer Portal. Since DXP 7.1 upgrades to Bootstrap 4 (we use Bootstrap 4.4.1 in Liferay 7.3), there are CSS style deprecations and removals that need to be executed. A guide to upgrading your theme from 6.2, 7.0, and 7.1 to 7.2 can be found here. The final change to themes is the removal of Velocity templates. Velocity templates were deprecated in DXP 7.0 in favor of Freemarker templates. If your DXP 7.0 project uses Velocity, you must convert them to Freemarker.

Additional Resources

- Liferay DXP Upgrade Reference Guide — Read this first before planning and executing your upgrade.
- What's New in Liferay DXP 7.3 — Summary of the new and improved features in Liferay DXP 7.3.
- Liferay Developer Guide — Provides key tutorials for various stages of the code upgrade process. The below sections are extremely important to review and understand, especially when deciding when to convert WAR based plugins to OSGi modules.
  - Migrating from Plugins SDK to Liferay Workspace
  - Planning Upgrades and Optimizations for WAR based plugins
  - Upgrading Plugins
  - Upgrading Themes
  - Upgrading and Migrating From EXT
- Blade CLI — Command line tool for developing in Liferay DXP
- Liferay Blade Samples — Repository of sample code and use cases

Summary

An upgrade to Liferay DXP has the potential to unlock the full range of benefits in the latest version of the Liferay platform. But each company must determine for itself whether an upgrade will be beneficial for the direction of the business after carefully weighing the costs, risks, time frame, labor, and business benefits involved.

Moving Forward

Our Liferay Global Services team is ready to provide a deep analysis of your specific requirements, help you form a personalized upgrade plan and offer you inside knowledge on setting your company up for success with Liferay. Learn more about Liferay Digital Experience Platform and the consulting services available to you by contacting sales@liferay.com.

Liferay makes software that helps companies create digital experiences on web, mobile and connected devices. Our platform is open source, which makes it more reliable, innovative and secure. We try to leave a positive mark on the world through business and technology. Hundreds of organizations in financial services, healthcare, government, insurance, retail, manufacturing and multiple other industries use Liferay. Visit us at liferay.com.

210323