# Questions Regarding LGPL 2.1 in FOLIO

As part of the FOLIO Technical Council's process for reviewing new modules, the open source license of the submitted code and the dependencies of the submitted code are evaluated. A sub-group of the TC has been working on criteria for such reviews. This document lists questions from the sub-group that could benefit from **legal assistance** in answering or clarifying, and potentially **policy decisions** from a FOLIO governing authority. It is bringing these questions to the FOLIO Community Council for consideration.

The FOLIO project is licensed under the Apache 2.0 open-source license. Some of FOLIO's code uses dependencies that are licensed under the GNU LGPL (Lesser General Public License). Version 2.1 of the LGPL license is particularly problematic because it contains conditions that don't fit neatly within an Apache-2.0-only distribution. The Apache Software Foundation's 3rd Party License Policy explicitly excludes LGPL versions 2.1 and 3.0 in its guidance on acceptable licenses for inclusion of third-party Open Source components in Apache Software Foundation products. (See the section on Category X: What can we NOT include in an ASF Project.)

Some open source projects make special efforts to address this contradiction. The only path that is sometimes considered acceptable is to keep the LGPL 2.1 code completely separate and have the Apache-2.0 code load it at run-time (dynamic linking). Even then, it is suggested that:

- You must still make the LGPL component's source code available on request.
- You must ship the LGPL license text and preserve copyright notices.
- You cannot redistribute the LGPL component under Apache 2.0 terms.
- You must document that the combined work is subject to two licenses.

For the purposes of these questions, "primary project" refers to the FOLIO component that has transitive dependencies of concern. Additional details, including examples and other information, can be found at the end of this document.

1. Can we use a LGPL-licensed dependency in a FOLIO module?
2. Are there distinctions between direct dependencies and transitive dependencies with LGPL 2.1? In general?
3. Are there distinctions between back-end and front-end uses with the LGPL 2.1? In general?

4. Are there distinctions between using projects under one language with projects under another language with the LGPL 2.1?
5. For projects using “peer-dependency” that are neither packaged nor distributed with the project, are there any risks from using LGPL 2.1? In general?
6. For libraries being used only for testing, can LGPL 2.1 be used?
    a. Can these testing-only libraries be used if they **are not** being distributed?
    b. Can these testing-only libraries be used if they **are** being distributed?
7. Can Java JAR and WAR files be reasonably considered independent and separate works in themselves, specifically when it comes to transitive dependencies that have LGPL 2.1?
    a. (Under the explicit and specific assertion that the transitive LGPL 2.1 dependency  licensed library is not modified or altered in any way.)
8. Is downloading and directly compiling the source into a binary at startup within a Docker image considered distributing?
    a. (Under the explicit and specific assertion that the Docker image itself neither contains the sources nor the binaries when distributing the Docker image itself.)
9. Do we need to provide the source code for the transitive LGPL 2.1 dependency?
    a. As in, can we require the user to obtain their library on their own, such as via websites like GitHub or commands like mvn install and npm install?
10. Does linking to a transitive LGPL 2.1 dependency count as a "work that uses the Library" for cases where the primary project is linking to a direct dependency and has no direct references to the transitive dependency?
    a. (Even if that transitive dependency might be classified as a "work based on the library?)
11. Does a transitive dependency fall under the "mere aggregation of another work" from Section 2 of LGPL 2.1?
    a. Is the primary project considered part of a whole if the third-party dependency is in the same JAR or WAR file?
    b. (The context here is in consideration of section 2 on "These requirements apply to the modified work as a whole.".)
12. Are JAR files considered an “executable”?
    a. (In the context of how “executable” is used in the LGPL 2.1 license.)
13. Are WAR files considered an “executable”?
    a. (In the context of how “executable” is used in the LGPL 2.1 license.)
14. For languages that produce executable binaries (Go), can LGPL 2.1 license be used?
    a. (The Go language specifically creates executables.)

15. For languages where there are no binaries (Javascript, Typescript, Node, NPM), can we use LGPL 2.1?

## Additional Details and Examples

*These following sections are not answers to the referenced questions.*
*Instead, these provide additional details or examples associated with the questions above.*

### 1. Details Regarding: Can we use a LGPL-licensed dependency in a FOLIO module?

FOLIO is under the Apache 2.0 license.

In practice, it is common to have LGPL licensed packages depending on Apache licensed projects in languages like Java, such as dbunit. Some projects even add explicit exceptions to make the situation clear, such as the OpenVPN OpenSSL Exception.

### 2. Details Regarding: Are there distinctions between direct dependencies and transitive dependencies with LGPL 2.1? In General?

A **direct dependency** is *directly* called or accessed by a project.
A **transitive dependency**, also called a **nested dependency**, is *indirectly* called or accessed by a project.

For example, Okapi directly includes and uses the LGPL2.1 based cql-java.
For example, Spring Boot Starter, an Apache 2.0 licensed project, imports Jakarta Annotations API, a GPL/EPL2.0 dual licensed project. The **Jakarta Annotations API** is a transitive dependency to projects depending on **Spring Boot Starter**.

### 3. Details Regarding: Are there distinctions between back-end and front-end uses with the LGPL 2.1? In General?

The LGPL2.1 license makes no clear statements regarding the back-end or front-end uses.

However, the typical front-end project utilizes JavaScript, TypeScript, or other such non-compiled languages.

There is no linking performed, nor is there combining performed in such languages.
In these cases, it seems unclear how the LGPL2.1 applies.
However, the Node Package Manager (NPM) might include dependencies that are compiled in some manner, which may put them under the linking requirements of the LGPL 2.1.

The back-end is traditionally compiled languages, but back-end projects might also not be compiled.

## 4. Details Regarding: Are there distinctions between using projects under one language with projects under another language with the LGPL 2.1?

A C programming language compiled library could be linked to by a Java compiled project.

## 5. Details Regarding: For projects using “peer-dependency” that are neither packaged nor distributed with the project, are there any risks from using LGPL 2.1?

With the Node Package Manager (NPM), a peer-dependency describes specific versions of a dependency that a project is claimed to be compatible with.
The project does not directly depend on the peer dependency.
A peer dependency, if not provided via a *transitive dependency* will not be included.

An Apache 2.0 licensed project could include an LGPL2.1 licensed peer dependency without actually needing or using the LGPL2.1 licensed dependency.

## 6. Details Regarding: For libraries being used only for testing, can LGPL 2.1 be used?

Test-only dependencies provide useful tools for projects, such as operating unit tests.
These dependencies are usually neither linked to a project nor bundled with a package.

The following is an example that presents why this question is being asked.

The Pa11u CI is a test tool intended for Continuous Integration (CI) accessibility testing and is under the LGPLv3 only license.
***This is not the LGLv2.1 license, but it provides a close use case example.***
Attempting to use this library for unit testing brings up the question of whether it can be used or not.

## 7. Details Regarding: Can Java JAR and WAR files be reasonably considered independent and separate works in themselves, specifically when it comes to transitive dependencies that have LGPL 2.1?

The Java JAR and WAR files are a packaging, or bundling, of code.
The JAR and WAR files, however, are not generally considered “executable”.
Some projects might alter the state of the JAR or WAR such that these files can be considered “executable”.

There are specific, official, clarifications regarding Java under the LGPL licenses.

The LGPL2.1 license has specific language regarding JAR files.

However, a Java dependency will not be directly linked to nor called by the project.
That LGPL2.1 Java dependency might itself also not need to be distributed with the project.

## 8. Details Regarding: Is downloading and directly compiling the source into a binary at startup within a Docker image considered distributing?

A Docker image can fetch all of the required source code, including transitive dependencies that could potentially contain LGPL2.1 licensed code, on start-up.
These sources are not necessarily distributed by the same group distributing the Docker image.
The *binary* is, technically, not being distributed because it is being built on the system by some entity after the Docker image is distributed to that same entity.
Does this technical interpretation hold true in the legal domain?

## 9. Details Regarding: Do we need to provide the source code for the transitive LGPL 2.1 dependency?

The third-party sources are fetched via some central and publicly available location such as with NPM or Maven repositories.

## 10. Details Regarding: Does linking to a transitive LGPL 2.1 dependency count as a "work that uses the Library" for cases where the primary project is linking to a direct dependency and has no direct references to the transitive dependency?

A third-party dependency typically is not directly used by some project.
There are no references to the third party dependency in the project itself in this situation.
Instead, a direct dependency might utilize call or otherwise use the third-party dependency.

Example Situation of a transitive dependency with no Direct References:
“My Project” depends on “Direct A”.
The “Direct A” project depends on “Transitive B”.
Both “My Project” and “Direct A” are under the Apache 2.0 license.
“Transitive B” is under the LGPL2.1 license.
“My Project” does not directly reference or use “Transitive B”.
Therefore, “Direct A” could potentially be replaced with another project called “Direct C”.
“Direct C”, under Apache 2.0 license, implements “Direct A” without depending on “Transitive B”.
The existence, or lack thereof, of “Transitive B” is essentially irrelevant to “My Project”.

## 11. Details Regarding: Does a transitive dependency fall under the "mere aggregation of another work" from Section 2 of LGPL 2.1?

Consider the example from the previous question details (“My Project”, “Direct A”, “Indirect B”, and “Direct C”).

Does a situation like the one described in that example fall under the “mere aggregation of another work”?

## 12. Details Regarding: Are JAR files considered an “executable”?

The Java JAR and WAR files are a packaging, or bundling, of code.

The JAR and WAR files, however, are not generally considered “executable”.

Some projects might alter the state of the JAR or WAR such that these files can be considered “executable”.

These are technical distinctions and might not be legal distinctions.
The LGPL2.1 license has specific language regarding what an “*executable*” is.

## 13. Details Regarding: Are WAR files considered an “executable”?

The same situation as described in the previous details section, but specifically for a WAR file instead of a JAR file.

## 14. For languages that produce executable binaries (Go), can LGPL 2.1 license be used?

This question is broader than the Go language, but the Go language is both supported in FOLIO and provides an example of a language that operates around an “executable”.

The typical Go language project compiles its dependencies into an executable.
The LGPL2.1 has specific language and requirements regarding an “executable”.

The Go language uses the term “module” in place of a library.

That standard method of distributing a module in the Go language is by downloading the source code and compiling it into the resulting executable.
This might result in a “combined work” that is also an “executable” in the terms of the LGPL2.1 license.

## 15. Details Regarding: For languages where there are no binaries (Javascript, Typescript, Node, NPM), can we use LGPL 2.1?

Non-compiled languages that produce no binaries are not clearly defined and understood when it comes to the LGPL2.1 license.

This question is centered around the expectation that the “binary” and “executable” requirements of the LGPL2.1 license may not apply to languages that distinctly lack binaries.

It might be possible to consider some code written in these languages to be “executable” without being a “binary” (such as with scripts).

The LGPL2.1 license is explicitly primarily written for the C programming language.
The standard interpretation of “linked”, “executable”, and “binary” in the C programming language is both distinct and clear.
This makes asserting that “linking” and “executable” are referring to “binaries” seem straight forward from a technical perspective.
Based on the C style definitions, languages like Javascript might neither classify as “linking” nor as a “binary”.
The legal perspective of this assertion is very unclear due to insufficient language in the LGPL2.1 license.