References to Other Packages

Use references to access objects or get datasets from other packages. These objects include workflow nodes, connections, and derived components.

To make an object available in other packages, you must set its visibility to "public" using the visibility parameter.

Creating a reference

To create a reference to another package:

1. On the **Navigation** toolbar, select **References**.
2. In the context menu, select **Add**. The **Add reference to package** dialog opens.
3. In the **Package file path** field, enter the path to the package that you want to access.
4. Click **Add**.

After you create a reference to an external package, you can do the following:

- Use its objects without changing them, through such components as Reference node and Node execution.
- Use derived components created within it. This allows you to modify the components' internal logic.

Reference properties

A reference to package has the following properties:

- **Name**: The package's name (the package file name without extension, automatically formated in line with the valid identifier requirements).
- **Path**: The path to the referenced package. In server editions, this will be the path from the file storage root directory. In desktop editions, an absolute or relative file path can be specified.
- **Version**: The current version of the referenced package. The version property is used to check if the version of the package located at the specified path matches the version in the package reference. If they don't, Megaladata returns an error message.

Setting a package version

Set this property in the **Properties of reference to package** dialog. You can open it in the following ways:

- Select a reference in the reference list and click **Configure** on the toolbar; or
- Select **Configure** in the reference's context menu; or
- Double-click the reference in the main workspace.

The version property is set in form of a wildcard. By default, `^1.0.0` is used. You can set the version in the following ways:

- **Specify exact version number.** E.g., `1.2.3` or `1.0.0-beta`.1
- **Use the caret (^) symbol.** The wildcard will include the specified version and all the versions not above the first non-zero part of the wildcard. Examples:
  - `^1.2.3` for the version range `>=1.2.3 <2.0.0-0`
  - `^0.2.5` for the range `>=0.2.5 <0.3.0-0`
  - `^0.0.4` for the `0.0.4` version only.
  - `^1.2.3-beta.2` for `>=1.2.3-beta.2 <2.0.0-0`
  - `^0.0` for `>=0.0.0 <0.1.0-0`
  - `^0` for `>=0.0.0 <1.0.0-0`
- **Use the tilde character (~).** This will include the specified version and all the minor versions above it. Examples:
  - `~0`, `~1`, etc. will function the same way as a `^` wildcard.
  - `~1` will include the range `>=1.0.0 <2.0.0-0`
  - `~1.2` will include `>=1.2.0 <1.3.0-0`
  - `~1.2.3` for `>=1.2.3 <1.3.0-0`
  - `~1.2.3-beta.1` for `>=1.2.3-beta.1 <1.3.0-0`
  - `~0.2` for `>=0.2.0 <0.3.0-0`
  - `~0` for `>=0.0.0 <1.0.0-0`
- **Use the asterisk character (*).** The asterisk can be used in place of one of the numbers in the version to allow for the range in this part. Examples:
  - `*` for the range `>=0.0.0` (that is, includes any package version)
  - `1.*` for `>=1.0.0 <2.0.0-0`
  - `1.2.*` for `>=1.2.0 <1.3.0-0`

Shared packages

Consider the situation when you open a package that has a reference to another package (we'll call it "*external package*"). If that external package is not open in the current Megaladata session, it will be loaded as a *shared package*. You can see this in the **Session Manager** page on the **Administration** pane. The package will remain shared as long as at least one reference to it in other packages is active.

Important note: When you work with a **Reference Node** that references a node from another package, you need to remember that the necessary data from the shared package will be calculated and downloaded only once.

```
- All the employed nodes will be locked until the calculation is
through.
- If you request the data again, the previously obtained result will
be returned. That is, there will be no recalculation.
```

To force the data to be recalculated, you can perform any one of the following actions:

- Close all referencing packages: Close every package that refers to the external (shared) package.
- Update the external package: Change the creation date of the external package (for instance, by re-saving it). This action creates an updated copy in the shared packages.
- Open the external package: Open the necessary external package in your current session, which will prevent it from being loaded as "shared".

After any of these actions, you need to re-open your working package.

Example

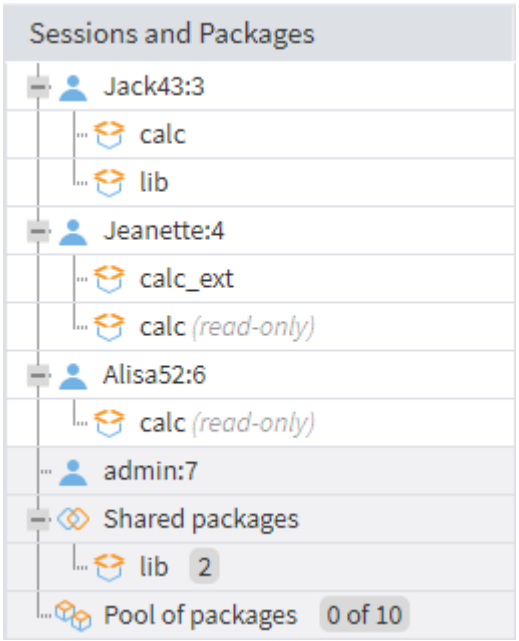


Figure 1. Working with an external package "lib"

The image demonstrates that several users have open packages that reference an external package `lib`.

The first user (Jack) has opened the external package `lib` and then his working package. In this case, the external package will access the data in the same session, regardless of the packages opened by other users.

The two other users (Jeanette and Alisa) have opened only their working packages under their sessions. This is why the external package `lib` is loaded as a shared package. (The count of shared packages at the bottom shows 2—the number of sessions referencing this package.) All the requests from these two users will be directed to the shared package. The results Alisa and Jeanette get in their Reference Nodes will be identical.

Service components

A referenced package might contain *service components*. These are recurring fragments intended for use only within that package.

Service component names are prefixed with a double underscore (__), and they usually appear nested under the main components. You cannot disable their visibility.

Read on: Annotations