ModusToolbox™ Library Manager user guide

ModusToolbox™ tools package version 3.7.0
Library Manager version 2.60

About this document

A newer version of this document may be available on the web here

Scope and purpose

This document provides information and instructions for how to use the ModusToolbox™ Library Manager.

Intended audience

Read this document to learn how to manage ModusToolbox™ BSPs and libraries for your application.

Document conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, menus and sub-menus
Italics	Denotes file names and paths.
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
File > New	Indicates that a cascading sub-menu opens when you select a menu item

Reference documents

Refer to the following documents for more information as needed:

- ModusToolbox™ tools package user guide

Table of contents

1 Overview

The Library Manager provides a GUI to select which Board Support Package (BSP) should be used when building a ModusToolbox™ application. The tool collects a list of available and currently selected BSPs and libraries, as well as all the necessary metadata from a webservice. The tool allows you to add and remove BSPs and libraries, as well as change their versions.

1.1 ModusToolbox™ 3.x vs. 2.x BSPs

One of the major changes made for ModusToolbox™ version 3.x is that most BSPs are no longer Git repos when you create a version 3.x application. Instead, the BSP becomes owned by the application. This means you can change various aspects of the BSP as you see fit without having to create a custom BSP, because creating a version 3.x application effectively creates a custom BSP. As such, you will typically check the BSP into source control with the application.

ModusToolbox™ version 2.x BSPs are still Git repos, by default, and may be used by more than one application in a given workspace. In general, this means any change you make to a BSP creates a "dirty" repo, and those changes affect all the related applications. In this case, you will typically not check the version 2.x BSP into source control, since you can regenerate the BSP. For this reason, it is best to create a "custom BSP" for your needs.

***Note**: BTSDK applications still default to shared BSPs, due to the nature of how BTSDK applications are structured and typical use cases.*

Refer to the ModusToolbox™ tools package user guide for more details about ModusToolbox™ BSPs and applications.

This version of the Library Manager supports both ModusToolbox™ version 3.x and 2.x BSPs. The main difference is that the **Version** and **Type** fields are not applicable and cannot be changed for a 3.x BSP. See the Properties tab section later in this document for descriptions of these fields.

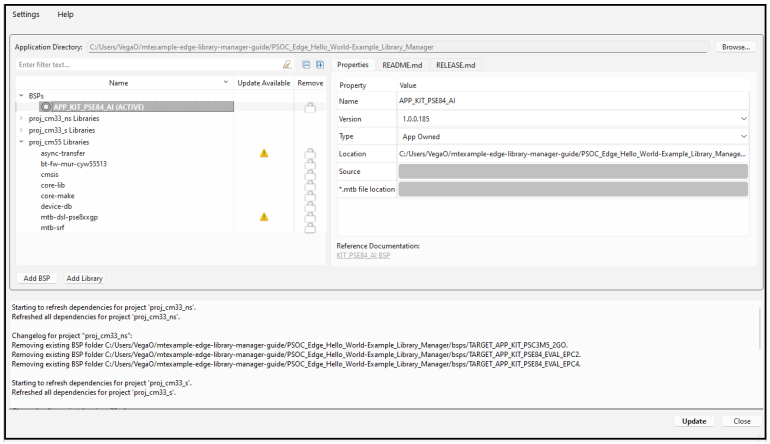
1.2 Specifying dependencies

This document only applies to applications that use .mtb files to specify their dependencies, not .lib files. To find information for applications that use .lib files, refer to an older version of this document.

Applications can share libraries. If needed, different applications can use different versions of the same library. Sharing resources reduces the number of files on your computer and speeds up subsequent application creation time. Shared libraries and versions are located in a "mtb_shared" directory adjacent to your application directories. You can easily switch a shared library to become local to a specific application, or back to being shared.

The Library Manager runs the make getlibs command to process the .mtb files, pull the libraries from the specified git repos, and store them in the specified location. The system also creates a file called mtb.mk in the application's libs subdirectory. The build system uses that file to find all the libraries required by the application.

The following shows the Library Manager for a typical PSOC™ Edge BSP. The key items are the lock symbols for libraries that are included because they are dependencies of other libraries. See Working with libraries for more details about using the tool.



The following concepts are key to understanding dependencies:

1.2.1 .mtb file

This file provides information about the associated BSP/library. It contains:

- A URL to a git repository somewhere that is accessible by your computer, such as GitHub.
- A git Commit Hash or Tag that tells which version of the library that you want.
- A path to where the library should be stored.

A typical *.mtb* file looks like this:

```
https://github.com/Infineon/core-make#release-v3.0.0#$$ASSET_REPO$$core-make/release-v3.0.0
```

The variable `$$ASSET_REPO$$` points to the root of the shared location. If you want a library to be local to the application instead of shared, use `$$LOCAL$$` instead of `$$ASSET_REPO$$`.

1.2.2 Direct dependency

This is a library that is directly referenced by an application. In some applications, there will be direct dependencies such as Wi-Fi or Bluetooth libraries. You may also have changed the location or version of an indirect library, which makes that library a direct dependency.

You manage the .mtb files for direct dependencies, and .mtb files should be checked into source control along with your application source code. A .mtb file for a direct dependency is typically stored in the application's deps subdirectory by default. You can store it anywhere inside the application except in the application's libs subdirectory.

1.2.3 Indirect dependency

This is a library that was included indirectly as a dependency of a BSP or another library. The system stores the code in the appropriate directory and resolves any potential conflicts with different library versions.

The system finds indirect dependencies for each library using information that is stored in a manifest file. For each indirect dependency found, the Library Manager places an *.mtb* file in the application's *libs* subdirectory. The tool also creates an *assetlocks.json* file in the application's *deps* subdirectory to record the locked release version of those libraries.

You do not manage *.mtb* files for indirect dependencies, and they do not need to be checked into source control. The *.mtb* file for an indirect dependency is stored in the application's *libs* subdirectory and must not be moved.

1.2.4 Shared

A shared library includes the code for the associated *.mtb* file, and the library can be shared by multiple applications in the same directory structure or workspace. When creating a new application, BSPs/libraries can be shared and placed in an *mtb_shared* directory adjacent to the application directory(ies), based on the "default_location" in the manifest file. When you add a BSP/library and specify it as shared, it is also included in the *mtb_shared* directory. These BSPs/libraries do not need to be checked into source control. All the code can be re-downloaded at any time from the information in the *.mtb* file.

You can change the name and location of the shared directory using make variables `CY_GETLIBS_SHARED_NAME` and `CY_GETLIBS_SHARED_PATH`. For more details about the ModusToolbox™ build system and make variables, refer to the ModusToolbox™ tools package user guide.

1.2.5 Local

A local library includes the code for the associated *.mtb* file, and the BSP/library is used only for a specific application. The local BSP/library is stored in the *libs* subdirectory of the specific application. These BSPs/libraries do not need to be checked into source control. All the code can be re-downloaded at any time from the information in the .mtb file.

If you change libraries to be local to an application (see Add library (single-core application) later in this guide), then the source code for these are located in the *libs* subdirectory inside the application directory.

1.3 Including libraries and dependencies

There are two basic ways of including a library and its dependencies into your application: using a manifest file or using manual methods. There is also a third method that is only used for custom BSPs. Each method is discussed separately below. The method you should use depends on whether or not the library and its dependencies are specified in a manifest file.

1.3.1 Including libraries that are specified in a manifest

All libraries provided by Infineon are specified in manifest files that are automatically found by the tools. Manifests are XML files that tell the Library Manager how to discover the list of available BSPs, libraries, and library dependencies. The manifest can also specify specific versions of libraries and dependent libraries that are meant to work together. For more information, refer to the Manifest chapter in the ModusToolbox™ tools package user guide.

You can create your own manifest file(s) for your libraries so that they will show up in the Library Manager and will work just like the Infineon libraries. More information on manifest files and how to create and use your own manifests is covered in the ModusToolbox™ tools package user guide.

In order to manage libraries that are specified in a manifest file in the application, files with the extension mtb are used. Their location and behavior depend on whether the library is a direct dependency or an indirect dependency. Normally the Library Manager is used to create/modify *mtb* files so you won't need to change them or even view them, but it is worthwhile to understand what is in them.

1.3.1.1 Direct dependencies

For direct dependencies, there will be one or more *mtb* files somewhere in your project (typically in the *deps* directory but could be anywhere except the *libs* directory). An *mtb* file is simply a text file with the extension mtb that has three fields separated by #:

- A URL to a Git repository somewhere that is accessible by your computer, such as GitHub

- A Git Commit Hash or Tag that tells which version of the library that you want
- A path to where the library should be stored in the shared location (that is, the directory path underneath *mtb_shared*).

A typical mtb file looks like this:

```
https://github.com/cypresssemiconductorco/retarget-io/#latest-v1.X#$$ASSET_REPO$$/retarget-io/
latest-v1.X
```

The variable `$$ASSET_REPO$$` points to the root of the shared location - it is specified in the application's *Makefile*. If you want a library to be local to the app instead of shared, you can use `$$LOCAL$$` instead of `$$ASSET_REPO$$` in the mtb file before downloading the libraries. Typically, the version is excluded from the path for local libraries since there can only be one local version used in a given application. Using the above example, a library local to the app would normally be specified like this:

```
https://github.com/cypresssemiconductorco/TARGET_CY8CKIT-062S2-43012/#latest-v1.X#$$LOCAL$$/
TARGET_CY8CKIT-062S2-43012
```

***Note**: The examples above specify dynamic library versions (e.g. latest-v1.X). Normally in your application, you will want fixed library versions (e.g. release-v1.0.0). This behavior can be controlled using the Library Manager.*

1.3.1.2 Indirect dependencies

For indirect dependencies, an *mtb* file for each library is automatically created in the *libs* directory. The version of each dependent library is also captured in the file *assetlocks.json* in the *deps* directory.

Once all the *mtb* files are in the application (both direct and indirect), the libraries they point to are pulled in from the specified Git repos and stored in the specified location (that is, *mtb_shared* for shared libraries and *libs* for local libraries). Finally, a file called *mtb.mk* is created in the application's *libs* directory. That file is what the build system uses to find all the libraries required by the application.

Since the libraries are all pulled in using `make getlibs`, you don't typically need to check them in to a revision control system - they can be recreated at any time by re-running `make getlibs`. This includes both shared libraries (in *mtb_shared*) and local libraries (in *libs*) - they all get pulled from GitHub when you run `make getlibs`.

You also don't need to check in the *mtb* files for indirect references and the *mtb.mk* file which are stored in the *libs* directory. In fact, the default *.gitignore* file in our code examples excludes the entire *libs* directory since you should not need to check in any files from that directory – they can be recreated at any time.

1.3.2 Including libraries are not specified in a manifest

If you a have your own custom libraries, you can create and include your own custom manifest files so that your libraries will show up in the Library Manager just like Infineon libraries. That method also allows you to specify the dependencies in the manifest so that they are automatically added as indirect dependencies when the dependent library is added to the application. Custom manifests are covered in the ModusToolbox™ tools package user guide.

1.3.2.1 Manually adding a library

If you do not want to create a manifest file for your custom libraries, then you must manually include the library and its dependencies in each application that uses the library. You can include the library several different ways, such as:

- git clone the version of the library that you want (or use some other revision control system) into the application's directory
- unzip an archive file of the library into the application's directory
- recursive copy the library into the application's directory
- modify the SEARCH variable in the application's *Makefile* to point to the library
- add an *mtb* file to the *deps* directory for the library and run `make getlibs` (requires that the library is in a Git repo)

For the first four methods, the library can go anywhere in the application's directory tree. Remember that if you put it in the *libs* directory it will not be checked into source control by default, so you will normally want put manually added libraries somewhere else.

If you use the *mtb* file (that is, the fifth method), there are two things to be aware of: the library will still not show up in the Library Manager; and latest locking will not be performed on that library so you should use a fixed version in the *mtb* file unless you want the dynamic update behavior.

1.3.2.2 Manually adding dependencies

You can easily add dependencies using the Library Manager tool (for dependencies that are in a manifest file such as Infineon libraries). For dependencies that are not in a manifest file, you can use the same manual methods that are used for including the library itself.

1.3.3 Specifying dependencies for custom BSPs

Custom BSPs use an alternate method to specify dependencies. The advantage is that the specification of the dependencies is self-contained within the custom BSP itself.

This method is accomplished by having files with the extension *mtbx* for each dependency in the *deps* directory inside the custom BSP. An *mtbx* file has the exact same content at an *mtb* file with a different file extension.

The process of getting dependencies from *mtbx* files only searches in the target BSP for the application, so if your application contains multiple custom BSPs, only the dependencies for the active target (as specified in the TARGET variable in the *Makefile* or on the command line) will be included in the application.

The *mtbx* files are automatically created for you, so you typically don't need to do anything with them unless you want to add or remove dependencies from your custom BSP. You can use the BSP Assistant to do this; for details, refer to the BSP Assistant user guide.

During `make getlibs`, *mtbx* files are treated just like indirect dependencies that are specified in a manifest file. That is, the process will copy each *mtbx* file to the application's *libs* directory and will lock the version in the *assetlocks.json* file in the *deps* directory. The libraries are then pulled down according to the information contained in the *mtb* files.

2 Launch the Library Manager

There are numerous ways to launch the Library Manager, and that depends on how you use the various tools included with ModusToolbox™ software.

2.1 make command

As described in the ModusToolbox™ tools package user guide build system chapter, you can run numerous make commands in the application directory, such as launching the Library Manager. After you have created a ModusToolbox™ application, navigate to the application directory and type the following command in the appropriate bash terminal window:

```
make library-manager
```

This command opens the Library Manager GUI for the specific application in which you are working.

2.2 VS Code and Eclipse

VS Code and Eclipse have tools to launch the Library Manager from within an open application. Refer to the applicable user guide for more details:

- VS Code for ModusToolbox™ user guide
- Eclipse IDE for ModusToolbox™ user guide

2.3 Stand-alone GUI mode

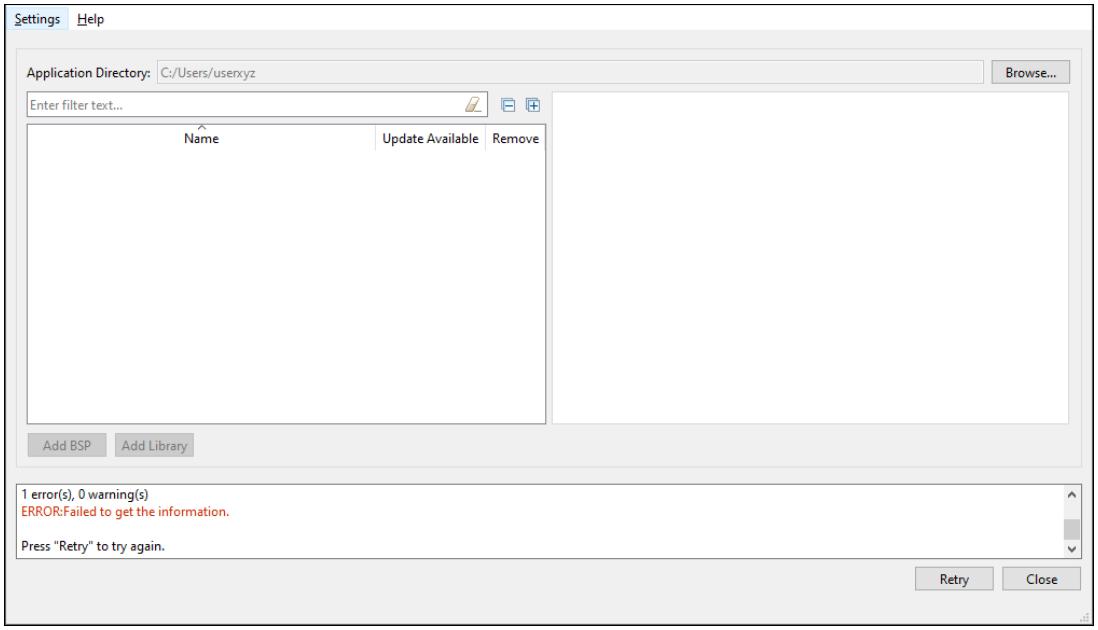
You can launch the Library Manager in stand-alone mode by running its executable as applicable for your operating system (for example, double-click or select it using the Windows **Start** menu). By default, it is installed here:

<install_dir>/ModusToolbox/tools_<version>/library-manager

When run in stand-alone mode, the Library Manager opens with the target directory set as <user-home> or as the directory selected from a previous stand-alone session.

***Note**: When launching from the executable, you can add the `--verbose [0-3]` option to specify more or less messaging in the console. The default level is 1.*

If you haven't opened the Library Manager tool previously, it starts at your home directory and looks for an application *Makefile*, and then it opens the first application it finds. If the tool does not find an application, it doesn't display any BSPs or libraries.



Click the **Browse…** button next to the **Application Directory** field and navigate to the appropriate directory that contains the desired application.

***Note**: The next time you open the Library Manager in stand-alone mode, it will open with the most recently selected application.*

2.4 Local Content mode

If the Library Manager cannot connect to the internet when launching, it displays messages/errors in the console that it cannot access the online manifest file. If you intend to work without Internet, enable **Local Content** mode from the **Settings** menu. Refer to the Local Content Storage CLI user guide for details about creating local content.

If you did not intend to work without Internet, check your Internet settings, adjust your proxy settings (from the **Settings** menu), and then click the **Retry** button to re-read library information.

2.5 Non-GUI command line interface (CLI)

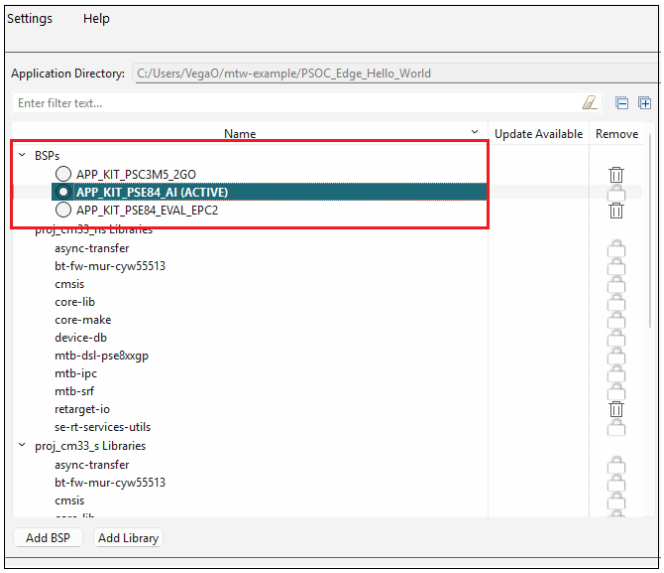
You can run a non-graphical interface from the command line. However, there are only a few reasons to do this in practice. The primary use case would be part of an overall build script for the entire application. For information about command line options, run the `library-manager-cli` executable using the `-h` option.

3 Working with BSPs

This section covers the common tasks involved when working with BSPs for your application, including:

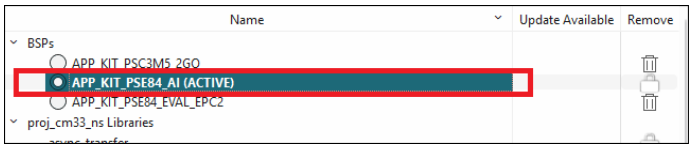
- Select Active BSP
- Add BSPs
- Rename BSP
- Remove BSP

The Library Manager displays the BSP(s) included with the application at the top of the list above the libraries, by default.



3.1 Select Active BSP

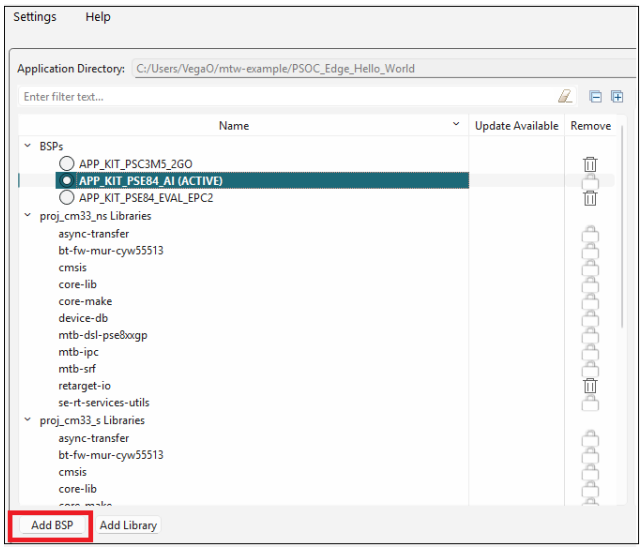
The active BSP for the application displays in bold text with "(ACTIVE)" next to it. An application can only have one active BSP. If your application has more than one BSP included, select one of the other BSPs to be active by clicking the radio option next to it. You can also choose to right-click on it and select Set as **Active BSP**. Then, click the Update button.



When you click the **Update** button, the system displays progress in the message console and updates your application's *Makefile* `TARGET=` variable to specify the new BSP as active.

3.2 Add BSPs

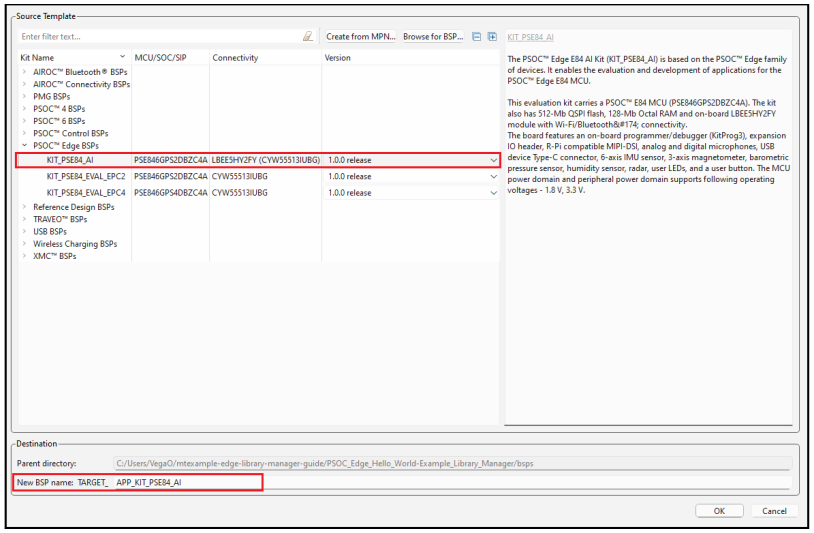
When you add a BSP to an application, you have several options, including adding a BSP from a template, browsing to one on disk, and creating a new BSP from a device part number. To begin the process, click the **Add BSP** button located below the list of BSPs and libraries.



This opens the Add BSP dialog. There are different steps to add, select, or create a BSP.

3.2.1 Add BSP from template

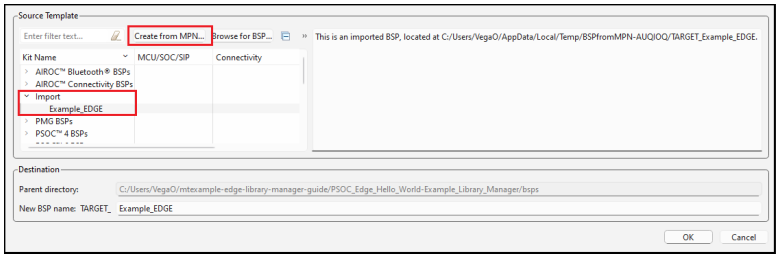
On the Add BSP dialog, select the desired BSP from the list, change the **New BSP name**, if desired, and click **OK**.



The added BSP will appear on the list of BSPs on the main Library Manager window.

3.2.2 Create BSP from MPN

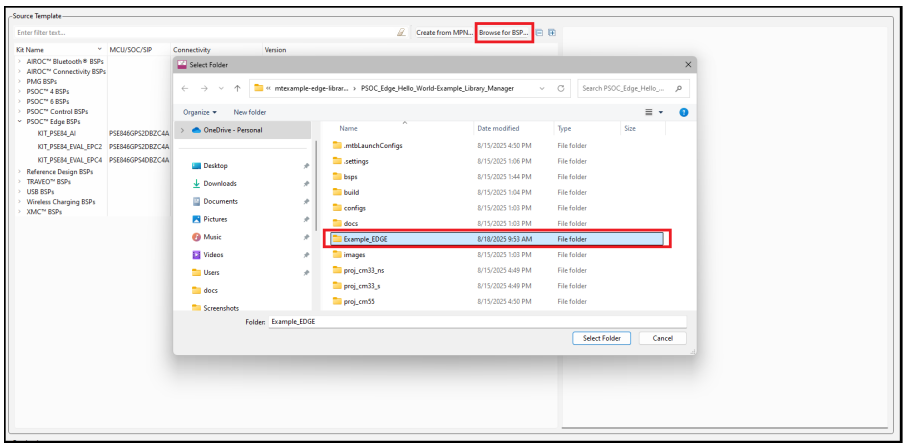
On the Add BSP dialog, click the **Create from MPN** button. This command opens the BSP Assistant tool to create a BSP based on device part numbers. Refer to the BSP Assistant user guide for more details about how to create a BSP from MPN. When you close the BSP Assistant, the new BSP will show up in the **Add BSP** dialog under Import.



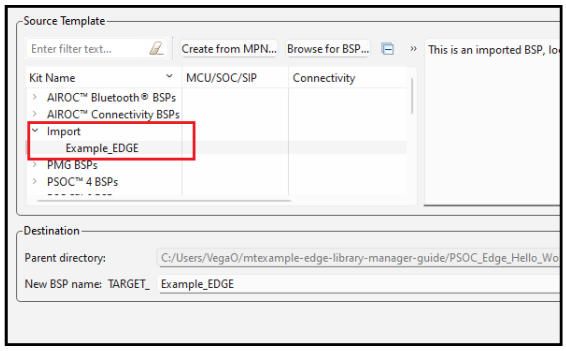
Change the **New BSP name**, if desired, and click **OK**. The new BSP will appear on the list of BSPs on the main Library Manager window.

3.2.3 Browse for existing BSP

On the Add BSP dialog, click the **Browse for BSP** button. Then on the Select Folder dialog, navigate to and select the desired BSP and click **Select Folder**.



The added BSP will show up in the Add BSP dialog under **Import**.



Change the **New BSP name**, if desired, and click **OK**. The added BSP will appear on the list of BSPs on the main Library Manager window.

3.2.4 Finalize Add BSP process

After adding a BSP you can choose to Select Active BSP. You can also choose to Remove BSP the non-active BSP and/or Rename BSP either BSP.

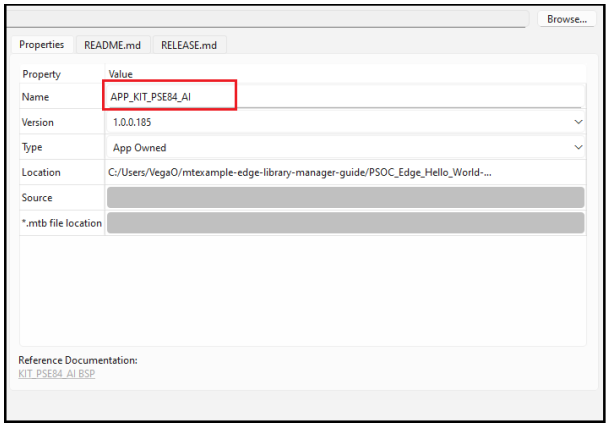
Regardless of whether you added an existing or new BSP, and whether you made other changes, the final step is to click the **Update** button on the main Library Manager window to commit the changes. This copies the BSP to the application and then it becomes owned by the application. This also updates the application's *Makefile*.

3.3 Rename BSP

There are a few ways to rename a BSP, as follows:

3.3.1 Existing BSP

For a BSP already shown in the main Library Manager list, right-click and select **Rename BSP** or double-click on the BSP. Then in the **Properties** tab Value field for the **Name**, type the new name for the BSP.

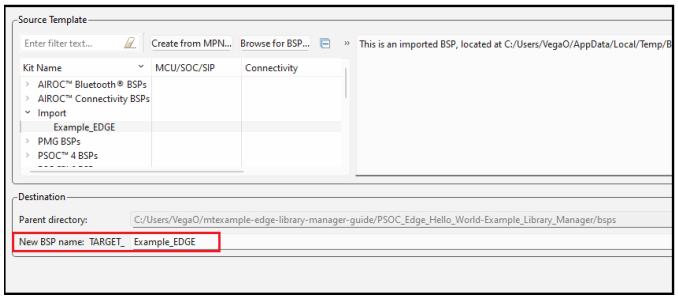


You can also double-click the BSP name shown in the **Value** field to rename it.

To commit the change, click **Update**.

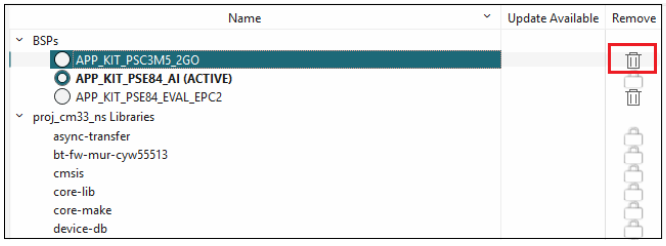
3.3.2 Added BSP

You can also rename a BSP when adding one. On the Add BSP dialog, the added BSP appears under **Import**. When you select the added BSP, you can type a new name for it in the **New BSP name** field.



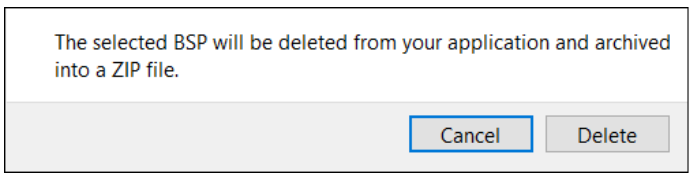
3.4 Remove BSP

To remove a BSP, click the **Delete** symbol under **Remove** for the appropriate BSP.



***Note**: The Lock symbol for the active BSP means you cannot remove the active BSP from the application.*

A dialog displays a message asking you to confirm the removal.



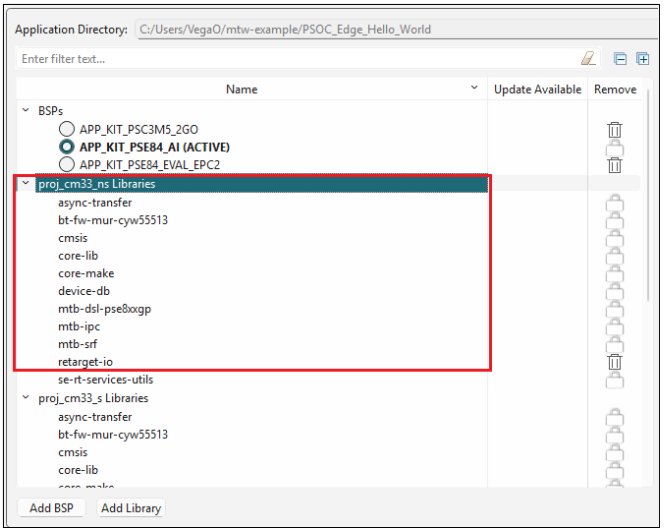
Click **Delete** to remove the BSP from the list of BSPs for the application. Then, click the **Update** button in the main window to commit the change.

4 Working with libraries

This section covers various ways to update BSPs and libraries, including:

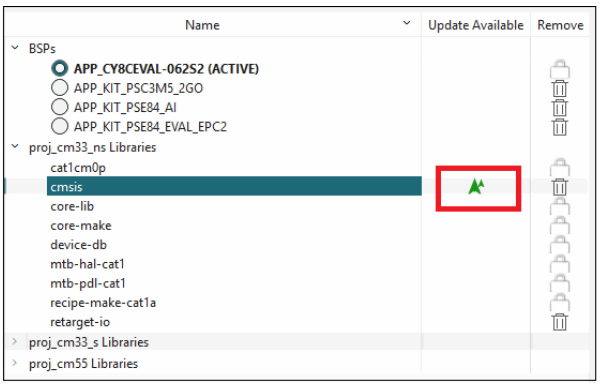
- Add library (single-core application)
- Add library (multi-core application)
- Update indirect dependency libraries
- Change library version
- Share/unshare libraries

The Library Manager displays the libraries included with the application below the BSP(s), by default.

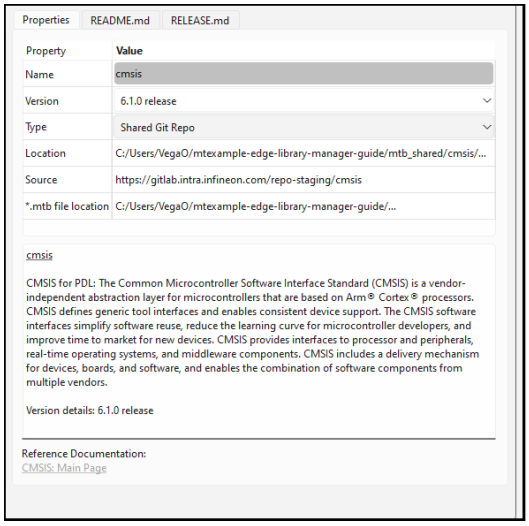


For a multi-core application, there are multiple sets of libraries; one for each core project.

When a library has a newer version available than the one currently selected, a "newer version" symbol displays under **Update Available**, indicating that a newer major or minor version is available.



When you select a library, various information displays on the right side of the Library Manager See Tabs for more information.

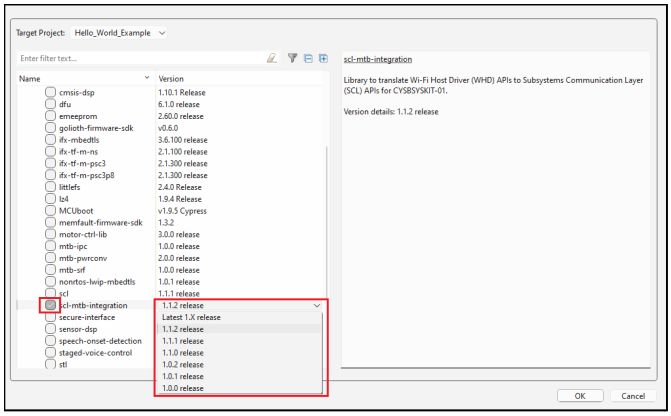


4.1 Add library (single-core application)

Click the **Add Library** button.

On the Add Library dialog, click one or more check boxes for libraries to add, and if needed, specify the **Version**. Then click **OK**.

***Note**: The **Target Project** option for a single-core application has only one choice and it is selected by default.*

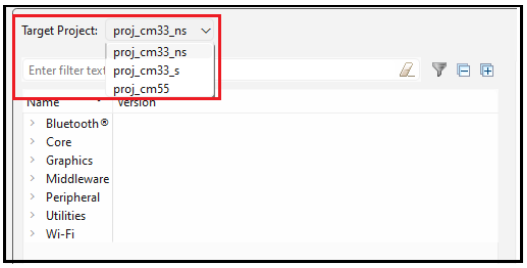


Back on the Library Manager, click the **Update** button to commit the changes.

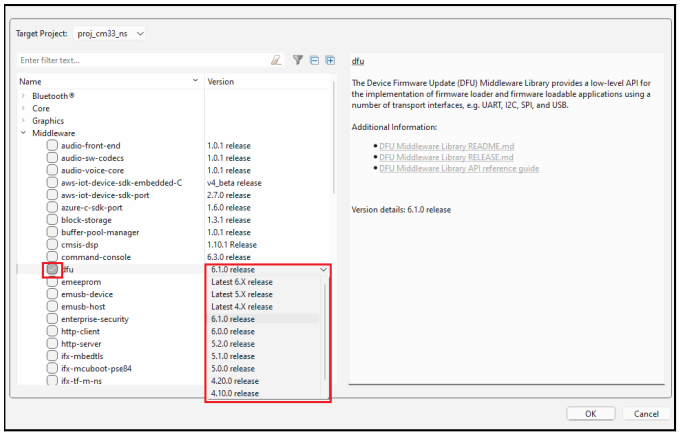
4.2 Add library (multi-core application)

Click the **Add Library** button.

Select the appropriate core project from the **Target Project** pull-down menu.

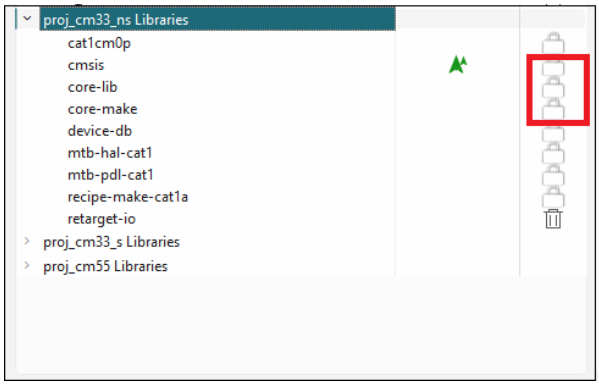


Click one or more check boxes for libraries to add, and if needed, specify the **Version**. Then click **OK**.



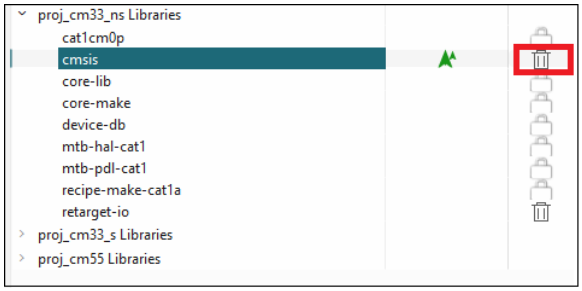
4.3 Update indirect dependency libraries

Some libraries included in your application show a lock symbol by default. This symbol indicates that a library is an Indirect dependency , and it is required by the BSP and/or another library.



These libraries cannot be removed using the Library Manager; however, you can change whether or not it is shared and select a different version.

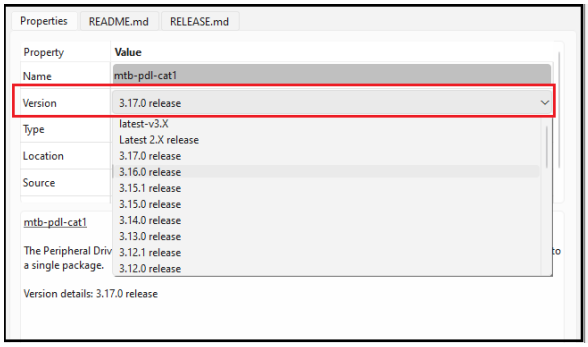
When you make a change to an Indirect Dependency library, the lock symbol changes to a **Delete** symbol. Unlike BSPs, this symbol indicates that after you click **Update**, the library is now a Direct dependency , and it is your responsibility to manage it either as a local library and/or with the selected version.



To change the library back to an Indirect Dependency, click the **Delete** symbol. The icon reverts back to the lock symbol, and the settings revert back to the defaults. You must click the **Update** button to commit the changes.

4.4 Change library version

Libraries have a specific version, by default. You can change a library version by selecting the pull-down menu next to **Version** and choosing another version. See Version for a detailed description.



Click the **Update** button to commit the changes.

When you change versions for a shared library, the tool creates a new subdirectory for the added version to ensure that the existing version is available for other applications. However, when you change a version for a local library, the system will not create a new local subdirectory. Instead, the version of the library will just be updated with the appropriate tag.

4.5 Share/unshare libraries

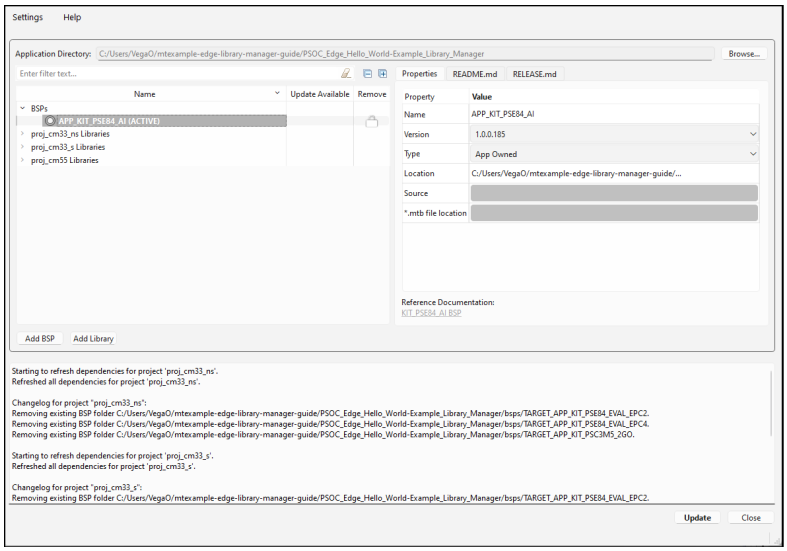
When you create applications, libraries can be shared or local depending on the application and manifest. Shared libraries are included in a *mtb_shared* directory, which is adjacent to your application directory. Libraries that are local (that is, not shared) are included in the *libs* subdirectory inside your application's directory.

If you select Type "Local Git Repo" under the **Properties** tab, a new library will be created in the libs subdirectory, and it will not be removed from the *mtb_shared* library. This is because the system assumes a shared library is being used by another application. However, if you select "Shared Git repo" for a library already included as local in the application and click **Update**, that library will be removed from the *libs* subdirectory and will instead be populated in the *mtb_shared* directory.

Click the **Update** button to commit the change(s). The message console displays the progress and indicates when changes are complete.

5 GUI description

The Library Manager contains menus, controls, tabs and buttons used to manage BSPs and libraries.



5.1 Menus

The Library Manager has two menus, as follows:

- **Settings** –
  - **ModusToolbox™ Settings:**: This opens the Settings tool, an editor that allows you to configure a wide range of settings for your environment, such as proxy settings, content modes, and manifest DB settings. See the Settings tool user guide for more details on specific features.
- **Help** –
  - **View Help**: Opens this document.
  - **About**: Displays tool version information, with a link to open Infineon.com.

5.2 Application directory

This is the location of the application's top-level directory, which contains one or more ModusToolbox™ Makefile projects. Use the **Browse...** button to select a different directory, if needed.

5.3 BSP/library controls

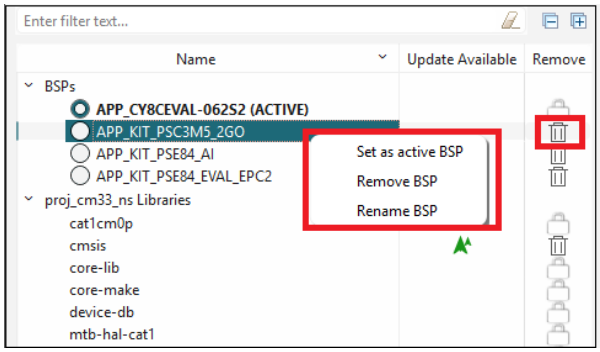
The left side of the Library Manager contains various controls for managing BSPs and libraries:

- **Filter text** – Field used to show only the BSPs/libraries that match the text entered.
- **Collapse All** – Click to collapse all item trees.
- **Expand All** – Click to expand all item trees.
- **SDK** – If you have selected a virtual SDK in Project Creator, it will appear here.
- **SDK Version** – If you have selected a virtual SDK, you can specify which SDK version you want to use here.
- **Remove Library** – This button allows you to remove libraries from the application.
- **Add BSP** – This button allows you to add and import BSPs to the application.
- **Add Library** – This button allows you to add libraries to the application.

***Note**: For more information about virtual SDKs, see the Project Creator user guide.*

5.4 BSP context menus

If you right-click on a BSP, different commands are available. See "Working with BSPs" for details.



- **Set as Active BSP** – This button allows you to switch between multiple BSPs, choosing which BSP is active and non-active.
- **Rename BSP** – This button allows you to specify the name for a BSP created as an in-app BSP.
- **Remove BSP** – This button allows you to delete a non-active BSP.

5.5 Tabs

The right side of the Library Manager contains the following tabs for a BSP or library: **Properties**, **README.md**, and **RELEASE.md**. Each tab may contain links to reference documentation that you can follow to find more specific information about your selected BSP or library.

5.5.1 Properties tab

The **Properties** tab contains several fields of information about the selected BSP or library.

5.5.1.1 Name

The **Name** field contains the BSP or library name. For BSPs, this field can be changed. See Rename BSP for more information on how to use the feature.

5.5.1.2 Version

The **Version** field doesn't apply to version 3.x BSPs, which are owned by the application. If this field is populated, it just displays information from the JSON file.

For all libraries and 2.x BSPs, you can select a dynamic "Latest X.Y release" or a fixed "X.Y.Z release" version. These represent tags for versions of libraries and version 2.x BSPs in GitHub repos:

- **Dynamic** – If you select a "Latest X.Y release " version, then it is dynamic. A ModusToolbox™ v2.x application will download and use the appropriate version specified in the manifest and attempt to resolve any potential conflicts with different versions. Using a "Latest" version means the item will update to the latest, backward-compatible version whenever you click the Library Manager **Update** button.
- **Fixed** – By selecting a specific "release-vX.Y.Z" version, you assign a fixed version of a specific, official release of the item that does not change, unless you manually change it.

When you create a new version 3.x application from an Infineon code example, the application converts libraries to use the "X.Y.Z release" version, by default. This ensures that they will not be updated automatically, unless you change the version to "Latest X.Y release."

When you open the Library Manager for an application and make any kind of change, the change summary displays in the console. If a "latest-vX.Y" tag has a newer version, the console displays a warning about the

"latest-vX.Y" tag. It includes a hyperlink to open a dialog that explains how the "latest-vX.Y" tag auto-update works.

If you click **Update**, all items with the "latest-vX.Y" tag will be moved to the newer version, even if you didn't make changes to them.

5.5.1.3 Type

The **Type** field doesn't apply to version 3.x BSPs. All 3.x BSPs are owned by the application.

For all libraries and 2.x BSPs, this field contains a pull-down menu to specify if the selected library is shared or local. See Specifying dependencies for descriptions of shared and local.

5.5.1.4 Location

This shows the current location on disk for the selected BSP or library.

5.5.1.5 Source

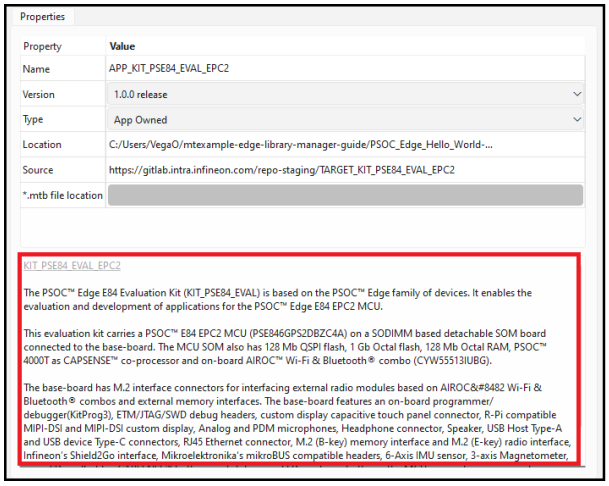
This field doesn't apply to version 3.x BSPs. It shows the source location (usually on GitHub) for the library or version 2.x BSP.

5.5.1.6 MTB file location

This field doesn't apply to version 3.x BSPs. It shows the location on disk for library and version 2.x BSP .mtb files.

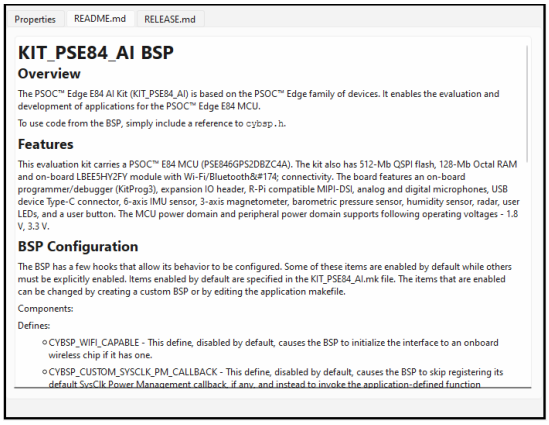
5.5.1.7 Description

For most libraries, there is a brief description included below the table.



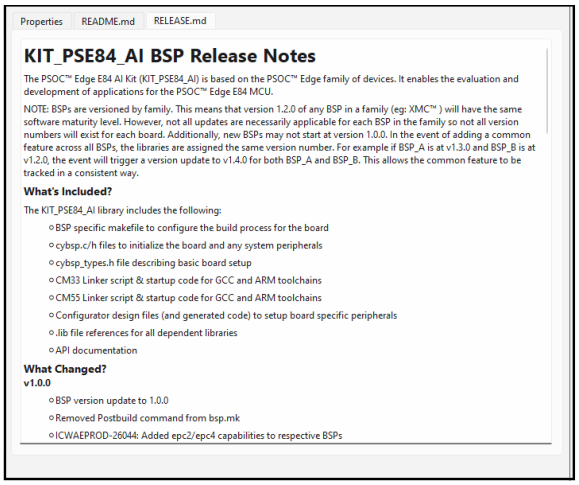
5.5.2 README.md tab

This tab displays the *README.md* file for the selected BSP or library, if available.



5.5.3 RELEASE.md tab

This tab displays the *RELEASE.md* file for the selected BSP or library, if available.



5.6 Buttons

The Library Manager contains the following buttons to perform the described actions:

- **Update** – Use this button to update your project with changes made in the Library Manager. This action runs the `make getlibs` command and you can use it at any time, even if you made no changes. You might do this for a project that has no libraries yet, or to update the libraries specified as "Latest" to get any updates. This button becomes bolded when there are pending changes requiring an update.
- **Close** – Use this button to close the Library Manager.
- **Retry** – This button displays if the message console indicates that the tool cannot access the manifest file. Use the **Retry** button after adjusting your internet and/or proxy settings to check if the tool can access the manifest file.

5.7 Message console

The area below the BSPs and Libraries displays various messages, such as when you select/deselect an item, click a button, or select Local Content mode.

6 Tool change description

This section lists and describes the changes for each version of this tool.

Version	Change descriptions
1.0	New tool.
1.1	Added **Settings** and **Help** menus. Moved link about version changes to the message console, when it is applicable. Added icon to indicate online/offline status. Removed Summary dialog; summary is shown in the console.
1.2.0	Added **Retry** button. Changed **Apply** button to **Update**. Added **Close** button. Tool can be launched from the Windows **Start** menu. Updated versioning to support patch releases. Updated for the MTB flow. Added toolbar commands to show/hide items, as well as collapse and expand the trees.
1.30	Added indications for newer versions of BSPs and libraries. For BSPs and libraries that are not selected, the displayed version was changed from Latest #.X release to the actual most recent X.Y.Z release.
1.40	Updated the handling on *.mtbx* files.
2.0	Entire GUI updated to support ModusToolbox™ version 3.0.
2.10	Added **Rename BSP** feature. Added **Create from MPN** feature. Changed Offline mode to Local Content mode. Removed online status icon.
2.20	Back-end changes and bug fixes. Updated the **Properties** tab to include descriptions.
2.21	Added Information Levels to **Settings** menu.
2.30	Added Manifest DB option to **Settings** menu.
2.40	Updated the **Settings** menu to add a link to the Settings tool and remove all of the items for general settings.
2.50	Back-end changes and bug fixes.
2.60	Back-end changes and bug fixes. Replaced X icon with garbage bin icon to more clearly indicate removal. Added **SDK** and **SDK Version** options in BSP/library controls.

Revision history

Revision	Date	Description
**	2019-10-16	New document.
*A	2019-10-17	Added a warning about removing all Cypress BSPs/libraries when there is no custom BSP.
*B	2020-03-26	Updated to version 1.1.
*C	2020-09-01	Updated to version 1.2.0.
*D	2020-10-07	Added details for BTSDK 2.8.
*E	2021-03-25	Updated to version 1.30.
*F	2021-09-22	Updated to version 1.40.
*G	2022-09-14	Updated entire document to reflect version 2.0.
*H	2023-05-09	Updated document to version 2.10.
*I	2024-02-12	Updated document to version 2.20.
*J	2024-09-27	Updated document to version 2.21.
*K	2024-12-06	Updated document to version 2.30.
*L	2025-03-21	Updated document to version 2.40.
*M	2025-08-29	Updated document to version 2.50.
*N	2025-12-12	Updated document to version 2.60.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-12-12

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-ugh1711392723157

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.