Please note that Cypress is an Infineon Technologies Company.

The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content

The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers

Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

ModusToolbox® Library Manager User Guide

Version 1.2.0

Overview

The Library Manager provides a GUI to select which Board Support Package (BSP) and version should be used by default when building a ModusToolbox application. The tool also allows you to add and remove libraries, as well as change their versions.

Beginning with the ModusToolbox 2.2 release, we've developed a new way of structuring applications, called the MTB flow. Using this flow, applications can share Board Support Packages (BSPs) and libraries. If needed, different applications can use different versions of the same BSP/library. Sharing resources reduces the number of files on your computer and speeds up subsequent application creation time. Shared BSPs, libraries, and versions are located in a new "*mtb_shared*" directory adjacent to your application directories. You can easily switch a shared BSP or library to become local to a specific application, or back to being shared.

Looking ahead, most example applications will use the new MTB flow. However, there are still various applications that use the previous flow, now called the LIB flow, and these applications generally do not share BSPs and libraries. ModusToolbox fully supports both flows, but it only supports one flow or the other for a given application. If an application uses the MTB flow, the system will process *.mtb* files only and ignore any *.lib* files.

Note When the BTSDK v2.8 is released, it will use the MTB flow.

Version

All BSPs and libraries have a version. Versions apply to both the MTB and LIB flows. See also BSP and Library Versions later in this document.

Dynamic

If a BSP or library uses the "Latest" tag, then it is dynamic. ModusToolbox will download and use the appropriate version specified in the manifest and any potential conflicts with different library versions. This uses the "Latest" tag that will update to the latest version whenever running the `make getlibs` command.

Fixed

By selecting a specific version from the pull-down menu in the Library Manager GUI, you assign a fixed version to the BSP or Library that does not change.

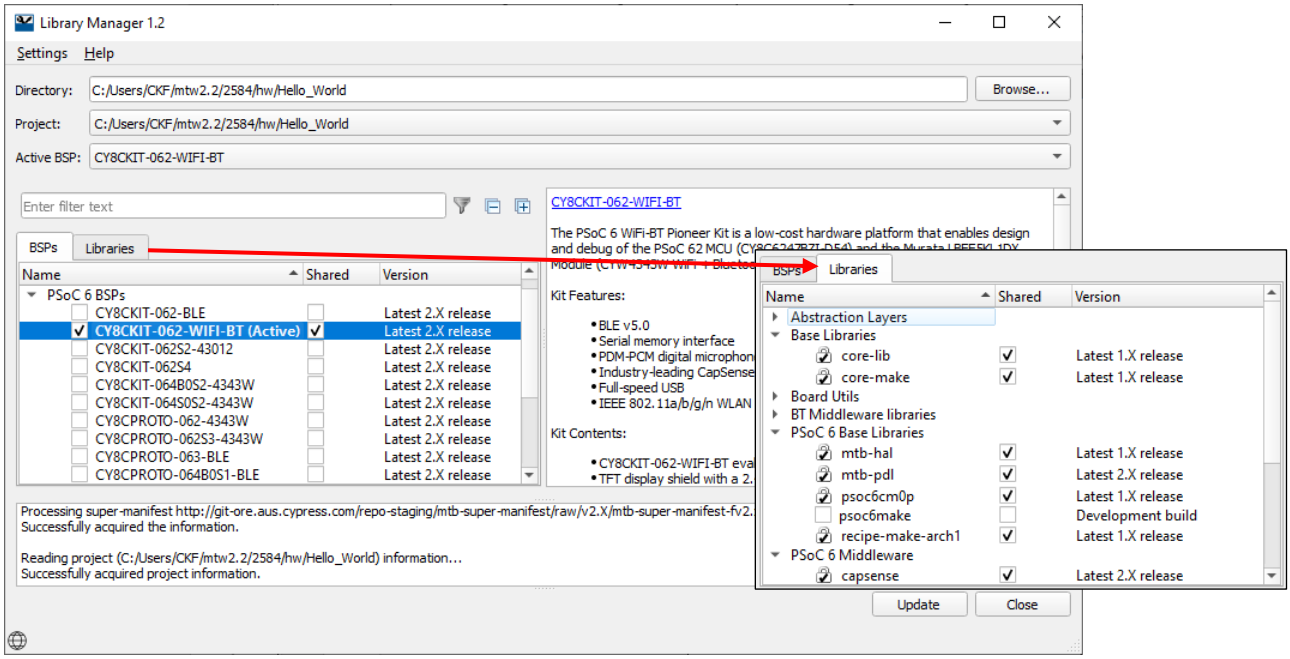
Manifests

Manifests are XML files that tell the Library Manager how to discover the list of available BSPs, libraries, and library dependencies. Manifests apply to both the MTB and LIB flows. For more information, refer to the Manifest chapter in the ModusToolbox User Guide.

MTB Flow

The MTB flow applies to ModusToolbox 2.2 and later releases. This flow uses *.mtb* files to manage libraries. 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 6 BSP and the MTB flow, with **BSPs** and **Libraries** tabs. The key items are the **Shared** column and the lock symbols for libraries that are included because they are dependendies of other libraries. See Working with BSPs/Libraries (MTB Flow) for more details about using the tool in the MTB flow.



The following concepts are key to understanding the MTB flow:

.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/cypresssemiconductorco/TARGET_CY8CKIT-062S2-43012/#latest-
v1.X#$$ASSET_REPO$$/TARGET_CY8CKIT-062S2-43012/latest-v1.X
```

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 `$$ASSET_REPO$$`.

Direct Dependency

This is a library or BSP that is directly referenced an application. In many applications, the only direct dependency is a BSP. In other applications, there will be additional 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.

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 a shared directory by default. It chooses the appropriate version specified in the manifest, as well as 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.

You do not manage these *.mtb* files, 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.

Shared

A shared BSP/library includes the code for the associated *.mtb* file, and the BSP/library can be shared by multiple applications in the same directory structure or workspace. By default, when creating a new application, all BSPs/libraries are shared and placed in an *mtb_shared* directory adjacent to the application directory(ies). 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.

In the MTB flow, the source code for all BSPs and libraries are placed in a shared directory named *mtb_shared* by default. Also by default, this shared directory is located adjacent to application directories in the same application root path. 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 User Guide.

Local

A local BSP/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.

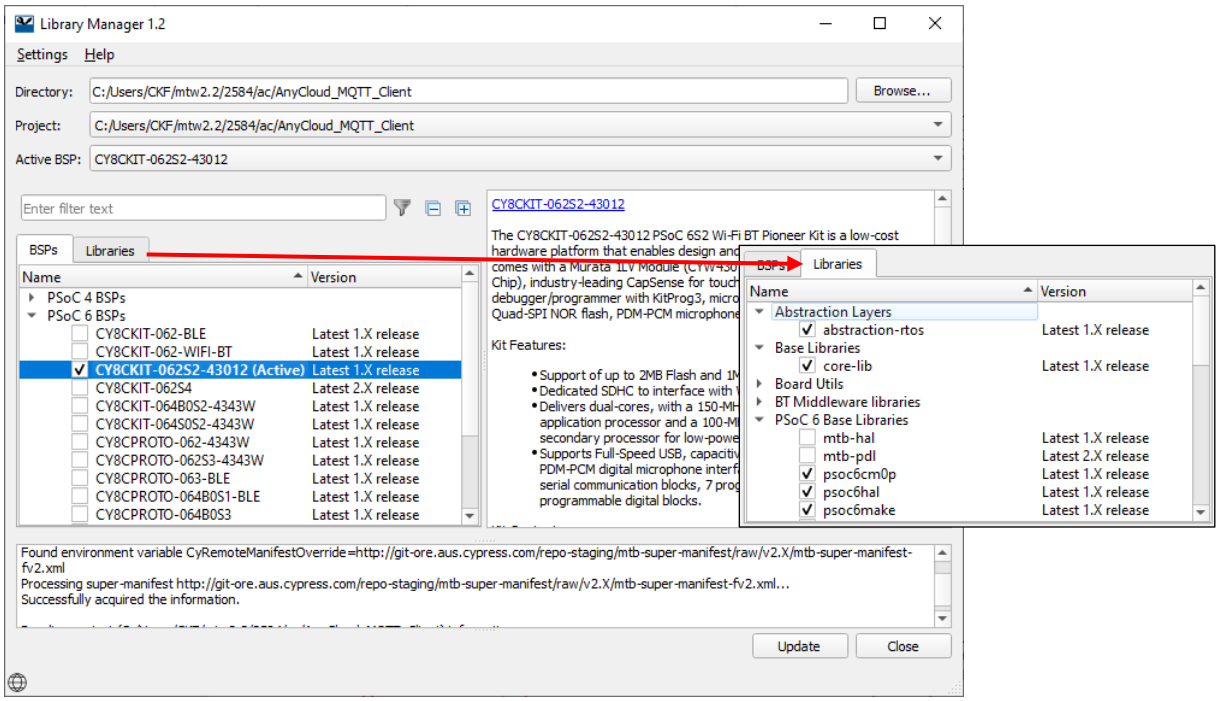
If you change BSPs and libraries to be local to an application (see Share/Unshare BSPs and Libraries later in this guide), then the source code for these are located in the *libs* subdirectory inside the application directory.

LIB Flow

At the time of the ModusToolbox 2.2 release, some PSoC 6 applications and the BTSDK v2.7 Bluetooth applications use the LIB flow, where libraries are managed with *.lib* files. These are some concepts for the LIB flow.

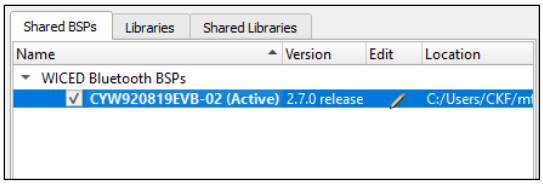
- **Local lib file –** This is a .lib file present in the project whose library is being edited.
- **Shared lib file –** This is a .lib file that is used indirectly via the shared library mechanism.
- **Target project –** This is the project directory whose library is being edited.
- **Fully editable mode –** You can add, remove, and change the version of the library.
- **Restricted edit mode –** You can manage the version of the selected library, but you cannot add or remove the library.

In the Library Manager, applications using the LIB flow also have the **BSPs** and **Libraries** tabs, but they do not have a **Shared** column and they do not distinguish between libraries that are included by the application vs. libraries that are included because other libraries depend on them. See Working with BSPs/Libraries (LIB Flow) for more details.



Note BTSDK v2.7 and older use the LIB flow, but they implemented a different approach to shared libraries before the MTB flow provided first class shared library support. In those versions, BTSDK projects rely on a separate "application" called 'wiced_btsdk' that is downloaded and installed next to the application root folder. This is not really an application, but rather a collection of the entire set of BTSDK BSPs and libraries in one structure. As such, there is no use case for the Library Manager with BTSDK LIB flow applications, since those applications contain all available BTSDK libraries.

Some applications also have **Shared BSPs** and **Shared Libraries** tabs.



Launch the Library Manager

You can launch the Library Manager from, and use it with, the Eclipse IDE for ModusToolbox. You can also run it without the Eclipse IDE. The tool collects a list of available BSPs and libraries for the selected project, as well as all the necessary metadata from a webservice. The tool also checks the current project directory for all *.mtb* or *.lib* files to find the currently selected libraries.

Note A single application can only use one flow. If the application uses the MTB flow, then all *.lib* files are ignored.

As a Stand-Alone Tool

You can launch the Library Manager as a stand-alone tool. By default, it is installed here:

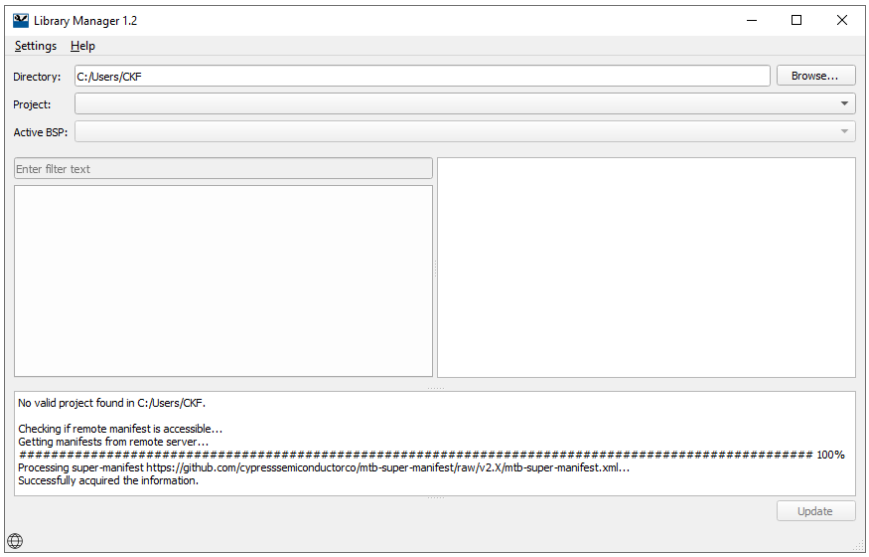
<install_dir>/ModusToolbox/tools_<version>/library-manager

On Windows, launch the tool from the **Start** menu. For other operating systems, navigate to the install location and run the executable.

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

If you are using the Library Manager in stand-alone mode, you must have a project that was created using `git clone` or the ModusToolbox Project Creator tool. The Project Creator tool runs the `make getlibs` command to prepare projects for the Library Manager to parse. If you manually create a project using `git clone`, you must either run `make getlibs` before launching the Library Manager, or click **Update** after launching the tool.

If you haven't opened the Library Manager tool before, it opens at your home directory and looks for directories with a makefile. It also scans for available manifest files to acquire BSP/library information.

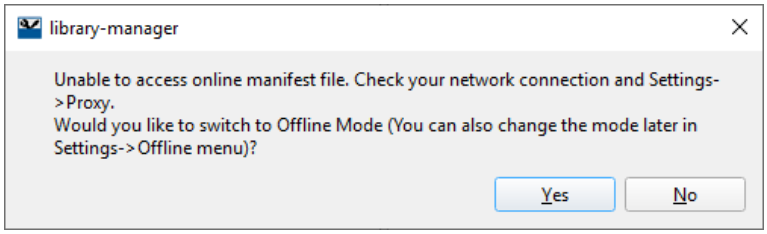


Note The next time you open the Library Manager, it will open with the most recently selected application.

Offline Mode

If the Library Manager cannot connect to the internet when launching, it displays a message in the console that it cannot access the manifest file. Adjust your internet and/or proxy settings (from the **Settings** menu), and then click the **Retry** button to re-read library information.

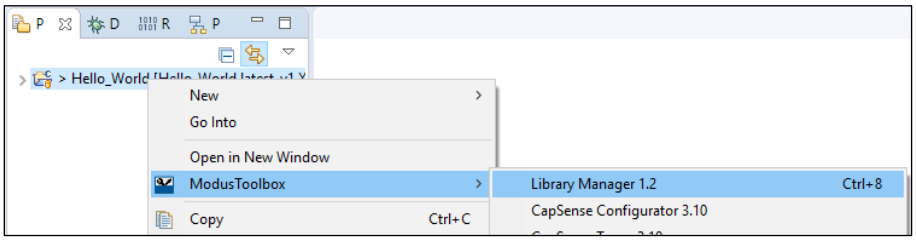
If you have the ModusToolbox Offline Content bundle installed and run the Library Manager without an internet connection, the following dialog displays:



If you click **Yes**, the Library Manager switches to offline mode and reads library information from the offline content bundle. If you click **No**, you can adjust internet and/or proxy settings, and then click **Retry** to re-read library information. Refer to the ModusToolbox User Guide for more details about the offline content bundle.

With the Eclipse IDE

To run the Library Manager from an application within the Eclipse IDE, right-click on the project and select **ModusToolbox > Library Manager**.



You can also open the Library Manager by clicking the link in the IDE Quick Panel. When opened from the IDE, the Library Manager points to the selected Eclipse workspace application directory. Refer to the Eclipse IDE for ModusToolbox User Guide for details about the IDE.

From the Command Line

You can run the tool 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.

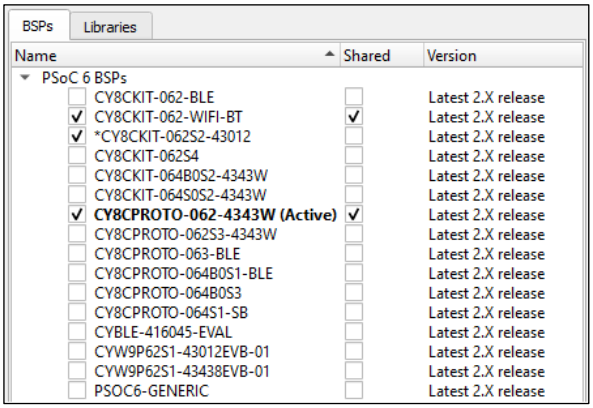
For information about command line options, run the tool using the `-h` option.

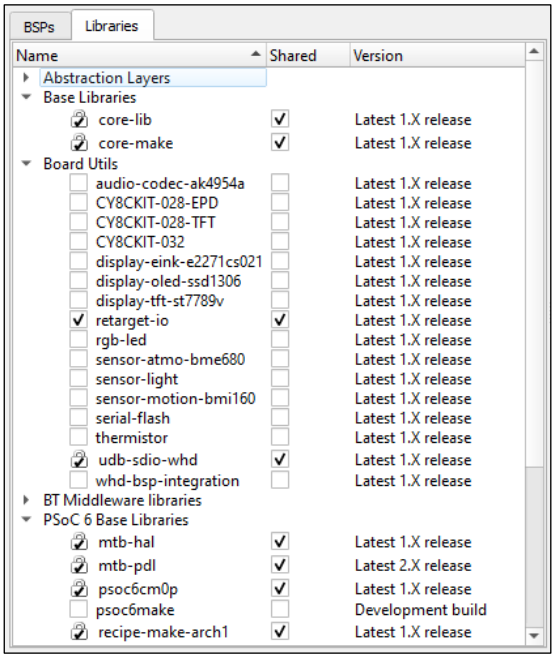
Working with BSPs/Libraries (MTB Flow)

This section covers various ways to update BSPs and libraries for the MTB flow, including:

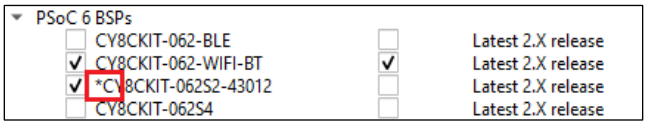
- Update Indirect Dependency libraries
- Add/remove BSPs or libraries
- Select Active BSP
- Share/Unshare BSPs and Libraries
- Change BSP/Library Version

When using the MTB flow, the **BSPs** and **Libraries** tabs show selection check boxes to the left of each of the listed BSPs and libraries. Selected check boxes represent BSPs/libraries included in the application, while those without selected check boxes are not included. Check boxes under the **Shared** column indicate if a BSP/library is shared or local, and each BSP/library shows a version under the **Version** column.



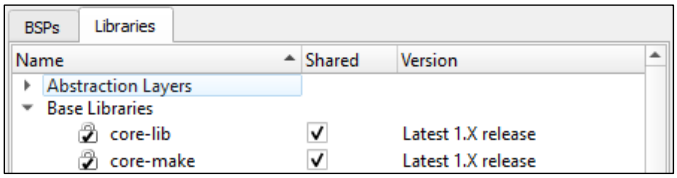


Note When you make a change to a BSP or library, an asterisk (*) displays next to it indicating a change is pending, but it has not yet been updated. Changes will take effect when you click the **Update** button.



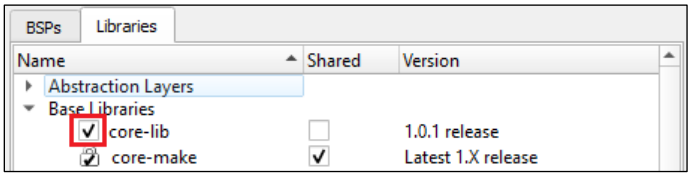
Update Indirect Dependency Libraries

Several 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.

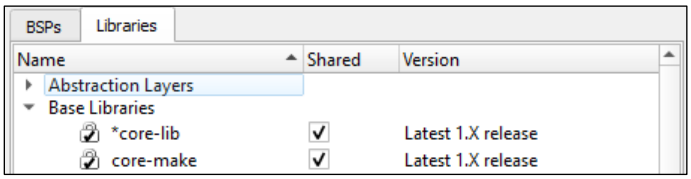


These libraries cannot be removed using the Library Manager; however, you can change whether it is shared or not, and you can select a different version.

When you make a change to an Indirect Dependency library, the lock symbol changes to regular check box. This 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.

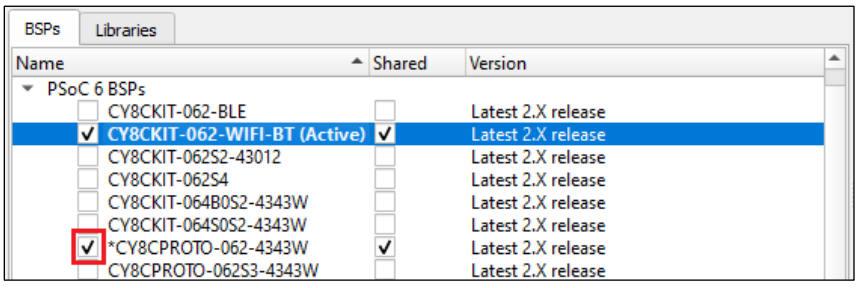


To change the library back to an Indirect Dependency, click on the check box. The icon reverts back to the lock symbol, and the **Shared** and **Version** settings revert back to the defaults. You must click the **Update** button to process the changes.



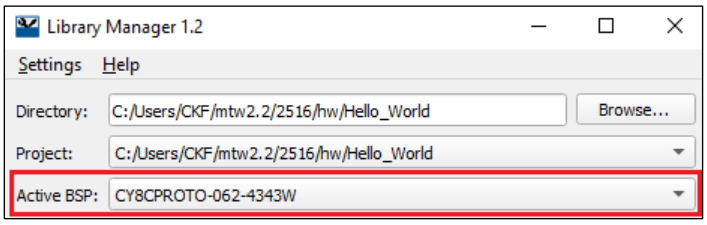
Add/Remove BSPs

Add BSPs to your application and remove them by selecting and deselecting the check boxes to the left of the BSPs. Click the **Update** button to process the change(s). The message console displays the progress and indicates when changes are complete.



Select Active BSP

The **Active BSP** for the application is shown in bold text with “(Active)” next to it. An application can only have one active BSP for which all settings apply. If your application has more than one BSP included (shown with selected check boxes), you can use the **Active BSP** pull-down menu to select one of the other BSPs to be active.

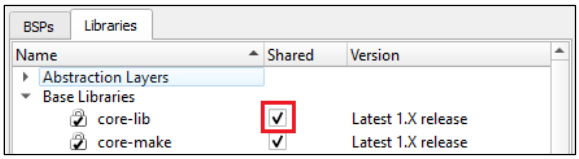


When you click the **Update** button, the system displays progress in the message console and updates your application's makefile `"TARGET="` field to specify the new BSP as active.

Note As of BTSDK v2.8, BTSDK applications only support one configured BSP at a time. Therefore, if you wish to change the Active BSP, you must deselect the previous BSP. This does not remove the BSP from the system; it only removes it from the application configuration and build search directories.

Share/Unshare BSPs and Libraries

When you create applications, BSPs and libraries provided by Cypress are shared automatically. By default, they are included in a *mtb_shared* directory, which is adjacent to your application directory. BSPs/Libraries that are local (aka, **not** shared) are included in the *libs* subdirectory inside your application’s directory.

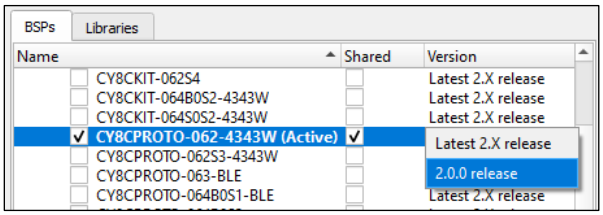


If you deselect the **Shared** check box, a new BSP/library will be created in the *libs* subdirectory, and it will **not** be removed from the *mtb_shared* library. This is because the system assumes a shared BSP/library is being used by another application. However, if you select the **Shared** check box for a BSP/library already included in the application and click **Update**, that BSP/ library will be moved from the *libs* subdirectory to the *mtb_shared* directory.

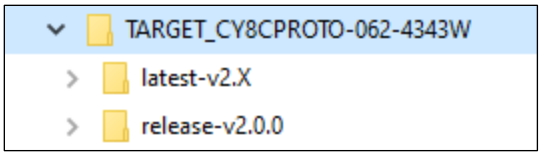
Click the **Update** button to process the change(s). The message console displays the progress and indicates when changes are complete.

Change BSP/Library Version

You can change a BSP/library version by selecting the pull-down menu under the **Version** column and choosing another version. See BSP and Library Versions for a detailed description.



When you change versions for a shared BSP/library, the tool creates a new subdirectory for the added version to ensure that the existing version is available for other applications. For example:



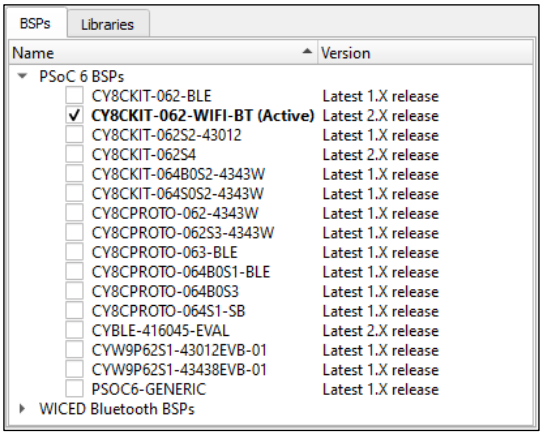
However, when you change a version for a local BSP/library, the system will **not** create a new local subdirectory. Instead, the version of the BSP/library will just be updated with the appropriate tag.

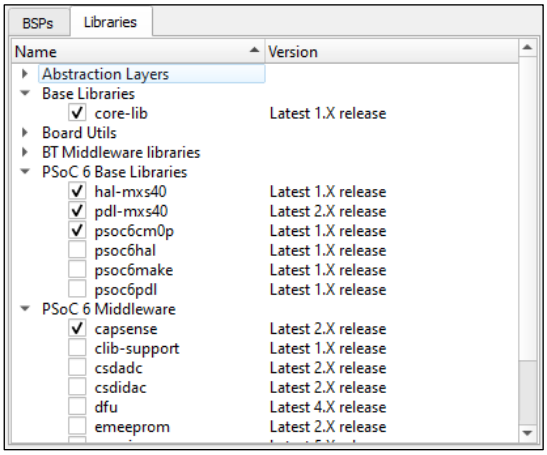
Working with BSPs/Libraries (LIB Flow)

This section covers various ways to update BSPs and libraries for the LIB flow, including:

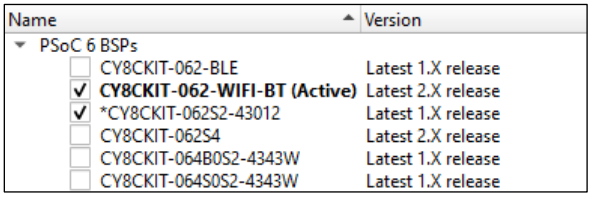
- Add BSPs/Libraries
- Remove BSPs/Libraries
- Update Shared BSPs/Libraries
- Change BSP/Library Version

When using the LIB flow, the **BSP** and **Libraries** tabs show selection check boxes next to each of the listed BSPs and libraries supported in the application. Selected check boxes represent BSPs/libraries already included in the application, while those without selected check boxes are not included. The Active BSP for the application is shown in bold text with “(Active)” next to it.





Note When you make a change to a BSP or library, an asterisk (*) displays next to it indicating a change is pending, but it has not yet been updated. Changes will take effect when you click the **Update** button.



Adding BSPs/Libraries

To add a BSP/library to your application, select the check box for one or more item on each tab and click **Update**. The tool starts updating the project and displaying progress in the status box.

```
Git is git version 2.17.0, found at /usr/bin/git

Searching application directory (.lib)...
Found 5 .lib file(s)
   Processing file "C:/Users/CKF/mtw2.2/2407/hw-2/CapSense_Buttons_and_Slider/deps/TARGET_CY8CKIT-062-WIFI-BT.lib"
warning: redirecting to http://git-ore.aus.cypress.com/repo-staging/TARGET_CY8CKIT-062-WIFI-BT.git/
   Processing file "C:/Users/CKF/mtw2.2/2407/hw-2/CapSense_Buttons_and_Slider/deps/TARGET_CY8CKIT-062S2-43012.lib"
warning: redirecting to http://git-ore.aus.cypress.com/repo-staging/TARGET_CY8CKIT-062S2-43012.git/
   Processing file "C:/Users/CKF/mtw2.2/2407/hw-2/CapSense_Buttons_and_Slider/deps/TARGET_CY8CPROTO-062-4343W.lib"
warning: redirecting to http://git-ore.aus.cypress.com/repo-staging/TARGET_CY8CPROTO-062-4343W.git/
```

When complete, the additional items will display with selected check boxes.

Remove BSPs/Libraries

To remove a BSP/library from your application, deselect the check box for one or more item on each tab, and click **Update**.

IMPORTANT NOTE: Use caution when removing all Cypress BSPs from your application. BSPs are responsible for ensuring key libraries are present, including "psoc6make" for PSoC 6 projects and “baselib” for Bluetooth projects. Before you remove all Cypress BSPs, make sure a custom BSP is present in your application. For information on creating and customizing BSPs, refer to the following:

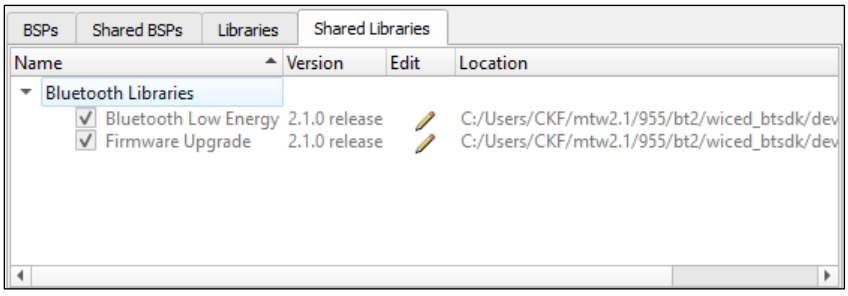
- **PSoC 6:** In the *ModusToolbox User Guide*.
- **Bluetooth:** In the *readme.md* file for the selected application.

Click **Update** and the tool starts updating the project and displaying progress in the status box.

When complete, the removed items will display in the Library Manager without selected check boxes. On disk, the libraries are removed by creating a special .lib file with a commit set to a sentinel value that indicates the library is removed.

Update Shared BSPs/Libraries

For certain applications, BSPs and Libraries are shared. In these cases, you may see additional tabs: **Shared BSPs** and **Shared Libraries**.



You cannot update items on these shared tabs directly. Instead, you must click the **Edit** icon . This opens another instance of the Library Manager where the BSP and Libraries are not shared. Edit the BSP/Library as described previously, and then close the second instance of the Library Manager.

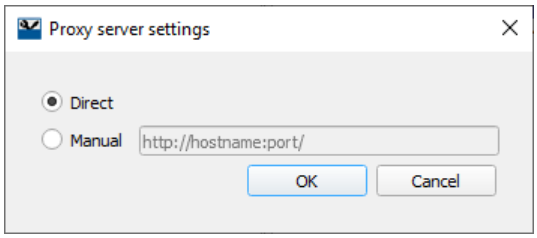
Note Even though the BSP/Library was updated in the 2nd instance of the Library Manager, the changes will not be displayed in the shared tabs until you close and reopen the 1st instance.

GUI Description

Menus

The Library Manager has two menus, as follows:

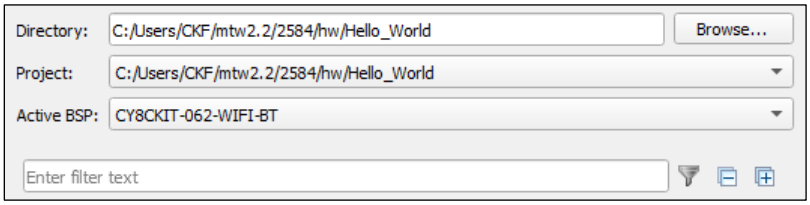
- **Settings** –
  - Offline: Check box to toggle between online and offline mode and read the local copy of the manifest file installed from the offline content bundle. See Offline Mode for more details.
  - Proxy Settings: Opens a dialog to specify direct or manual proxy settings.



- **Help** –
  - View Help: Opens this document.
  - About: Displays tool version information.

Fields

The Library Manager contains the following fields at the top of the GUI:



- **Directory** – 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.
- **Project** – Location of the application's selected makefile. Use the pull-down menu to select another project, if applicable.
- **Active BSP** – The name of the currently active target BSP. Use the pull-down menu to choose any of the boards with a selected check box in the **BSPs** tab. You can have multiple sets of BSP code in a project; however, the make project is built only for the selected **Active BSP**.
- **Filter text** – Field used to show only the BSPs/libraries that match the text entered.
- **Show/Hide** – Toggle to show only the BSPs/libraries with check boxes selected or show all BSPs/libraries. If there is filter text, then only BSPs/libraries with check boxes selected and BSPs/libraries with matching text will be shown instead of all.
- **Collapse All** – Click to collapse all item trees.
- **Expand All** – Click to expand all item trees.

Tabs

The Library Manager may display one or more of the following tabs, depending several variables:

- **BSPs tab** – Shows available BSPs and versions. This tab is hidden if there are no BSPs in the project and the project type is a library project or if the project uses shared BSPs.
- **Libraries tab** – Shows libraries supported by the Active BSP. This tab is hidden if the project type is a library project and if the project contains zero local libraries.
- **Shared BSPs tab (LIB Flow)** – Shows a read-only view of BSPs that are available via the LIB flow shared library mechanism. This tab is hidden if the project is not using any shared BSPs or if the project type is a library project.
- **Shared Libraries tab (LIB Flow)** – Shows a read-only view of LIB flow code libraries that will be used by the project. This tab is hidden if the project is not using any shared libraries or if the project type is a library project.

Shared Check Box (MTB Flow)

This column contains a check box that indicates if the associated BSP/Library is shared or not.

BSP and Library Versions

The Library Manager contains a **Version** column for each BSP and library shown in the tool. You can select "Latest X.Y release" or "X.Y.Z release." These represent tags for versions of BSPs/libraries in Cypress GitHub repos:

- The “latest-vX.Y” tag is updated on an ongoing basis to point to a newer, backward-compatible version of the library when Cypress releases it.
- The “release-vX.Y.Z” tag always points to a specific, official release, and it does not change.

When you create a new application from a Cypress code example, the application includes BSPs and libraries that use the “latest-vX.Y” version tag on the GitHub repo, by default. When Cypress releases a new version of a BSP/library, the “latest-vX.Y” tag on GitHub moves to point to that newer version.

The next time you open the Library Manager for that application and application and make any kind of change, the change summary displays in the console. If a remote 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 them.

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.
- **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.

Message Console

The area below the BSPs and Libraries displays various messages, such as when you select/deselect an item, click the **Update** or **Retry** button, or select Offline mode.

Status Indicator

At the bottom of the tool, the status indicator displays as faded in offline mode and displays as normal in online mode. If you hover over the indicator, a tooltip displays.

Manual Library Management

If a library is not in a manifest file, it will not appear in the Library Manager and will need to be managed manually. Regardless of the library management flow, this is a two-step process:

- Acquire the library
- Acquire the library's dependencies

These steps can be done several different ways depending on the construction of the library and the library management flow being used by the application. Note that for (custom) BSPs, the Project Creator tool simplifies this process greatly. You can still use the other methods on BSPs if you want to include a custom BSP into an existing application.

Note By manually managing libraries, all dependencies will be included directly in the application. Indirect dependencies only occur in the MTB flow when libraries and dependencies are included in manifest files as described in the Indirect Dependency section.

Acquiring a Library (MTB flow)

There are three basic ways to get a library that isn't in a manifest file using the MTB flow:

Requirements	Method
Library is hosted in a Git repo	Create a *.mtb* file for the library in the application's *deps* subdirectory.
	Run `make getlibs` from the application's root directory.
	The library can be local or shared based on the *.mtb* file contents.
Any library Library is local to app	Use copy, `git clone`, or use some other revision control system to pull the library into the application. It can be located anywhere except the application's *libs* subdirectory.
Any library Library is in a shared location	Use copy, `git clone`, or use some other revision control system to pull the library into a shared location. Edit the `SEARCH` variable in the application's Makefile to point to the library.
Custom BSP	Use copy, `git clone`, or use some other revision control system to get the BSP somewhere on disk if it isn't already. The location is not important. It will be copied into the application folder during project creation.
	Use the Project Creator **Import** function to select the BSP. The custom BSP will be copied into the application. See the Project Creator Guide for details.

Once you have a library, the next step is to get its dependencies. The methods used depend on whether you are using the MTB flow or the LIB flow.

Acquiring Dependencies (MTB flow)

There are several ways to acquire dependencies:

Requirements	Method
Library contains *.mtbx* files for its dependencies	Run `make import_deps` followed by `make getlibs` from the application's root directory (see import_deps for details).
Dependencies can be local or shared	
Library does not contain mtbx files but does contain lib files for its dependencies	Run `make lib2mtbx`, then `make import_deps` and finally `make getlibs` from the application's root directory (see lib2mtbx and import_deps for details).
Dependencies will be local	
Dependencies are in a manifest	Use the Library Manager.
Dependencies are hosted in git repos	Create a *.mtb* file for each dependency in the application's *deps* subdirectory.
	Run `make getlibs` from the application's root directory.
	The dependencies can be local or shared based on the *.mtb* file contents.
Any Dependency	Use copy, `git clone`, or use some other revision control system to pull the library into the application. It can be located anywhere except the application's *libs* subdirectory.
Dependency is local to app	
Any dependency Dependency is in a shared location	Use copy, `git clone`, or use some other revision control system to pull the library into a shared location. Edit the `SEARCH` variable in the application's Makefile to point to the library.
Custom BSP	Dependencies are pulled automatically by Project Creator.

import_deps

The `make import_deps` target is intended specifically to acquire dependencies in MTB flow applications for libraries that don't have their dependencies specified manifest files. To use it, you need a *.mtbx* file in the library for each dependency. The format of *.mtbx* files is identical to *.mtb* files. When you run `make import_deps`, it will copy the *.mtbx* files from the library into the application's deps folder while also changing the extension to *.mtb*. This results in the dependencies being direct to the application which are then pulled in when `make getlibs` is run. The libraries can either be local or shared depending on the contents of the *.mtbx* files. Usage:

```
make import_deps IMPORT_PATH=<path_to_library>
```

lib2mtbx

The `make lib2mtbx` target is mainly intended for libraries (or BSPs) that were created for ModusToolbox 2.0 or 2.1 that do not contain *.mtbx* files but do contain lib files for their dependencies. When you run `make lib2mtbx`, it will create *.mtbx* files in the library based on the *.lib* files in the library. The *.mtbx* files will be created to place the dependencies local to the application unless the optional argument "shared" is specified. You can save the *.mtbx* files to the library's source control so that you don't need to do this step the next time the library is included into an application. Usage:

```
make lib2mtbx CONVERSION_PATH=<path_to_library> [CONVERSION_TYPE="shared"]
```

Acquiring a Library (LIB flow)

There are two basic ways to get a library that isn't in a manifest file using the LIB flow.

Requirements	Method
Library is hosted in a git repo	Create a *.lib* file for the library in the application's *deps* subdirectory.
	Run `make getlibs` from the application's root directory.
Any library	Use copy, `git clone`, or use some other revision control system to pull the library into the application. It can be located anywhere except the application's *libs* subdirectory.
Custom BSP	Use copy, `git clone`, or use some other revision control system to get the BSP somewhere on disk if it isn't already. The location is not important. It will be copied into the application folder during project creation.
	Use the Project Creator **Import** function to select the BSP. The custom BSP will be copied into the application. See the Project Creator Guide for details.

Acquiring Dependencies (LIB flow)

Again, there are several ways to acquire dependencies depending on the library's structure:

Requirements	Method
Library contains *.lib* files for its dependencies	Run `make getlibs` (this step is already done if you used `make getlibs` to acquire the library).
Dependencies are listed in a manifest	Use the Library Manager.
Dependencies are hosted in a dit repo	Create a *.lib* file for each dependency in the application's *deps* subdirectory.
	Run `make getlibs` from the application's root directory.
Any dependency	Use copy, `git clone`, or use some other revision control system to pull the library into the application. It can be located anywhere except the *libs* directory.
Custom BSP	Dependencies are pulled automatically by Project Creator.

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.

© Cypress Semiconductor Corporation, 2019-2020. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, ModusToolbox, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.