ModusToolbox™

Software Overview

Document Number: 002-28029 Rev *A

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
http://www.cypress.com

© Cypress Semiconductor Corporation, 2019. 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.

Contents

1. Introduction

As a developer, you have preferred tools and workflows for creating products. What you need is enablement software you can use seamlessly in your existing environment. ModusToolbox™ software is a collection of tools and libraries that enable you to develop embedded and connected applications in your ecosystem of choice.

ModusToolbox software is:

- Comprehensive – it has the resources you need
- Flexible – you can use the resources in your own workflow
- Atomic – you can get just the resources you want

ModusToolbox software includes configuration tools, low-level drivers, middleware libraries, and operating system support, as well as other packages that enable you to create MCU and wireless applications. It also includes the optional ModusToolbox IDE. Unless specifically stated otherwise, ModusToolbox resources are compatible with Linux®, macOS®, and Windows®-hosted environments.

The ModusToolbox installer provides the core resources you need to get started, such as configurators that generate code based on your design. In addition, Cypress provides libraries and enablement software at the Cypress GitHub site. Some resources will be used by all developers. Others will be used by developers in particular ecosystems.

Cypress software resources available at GitHub support one or more of the target ecosystems:

- MCU and Bluetooth SOC ecosystem – a full-featured platform for PSoC 6, Wi-Fi, Bluetooth, and Bluetooth Low Energy application development
- Mbed OS ecosystem – provides an embedded operating system, transport security and cloud services to create connected embedded solutions
- Amazon FreeRTOS ecosystem – extends the FreeRTOS kernel with software libraries that make it easy to securely connect small, low-power devices to AWS cloud services

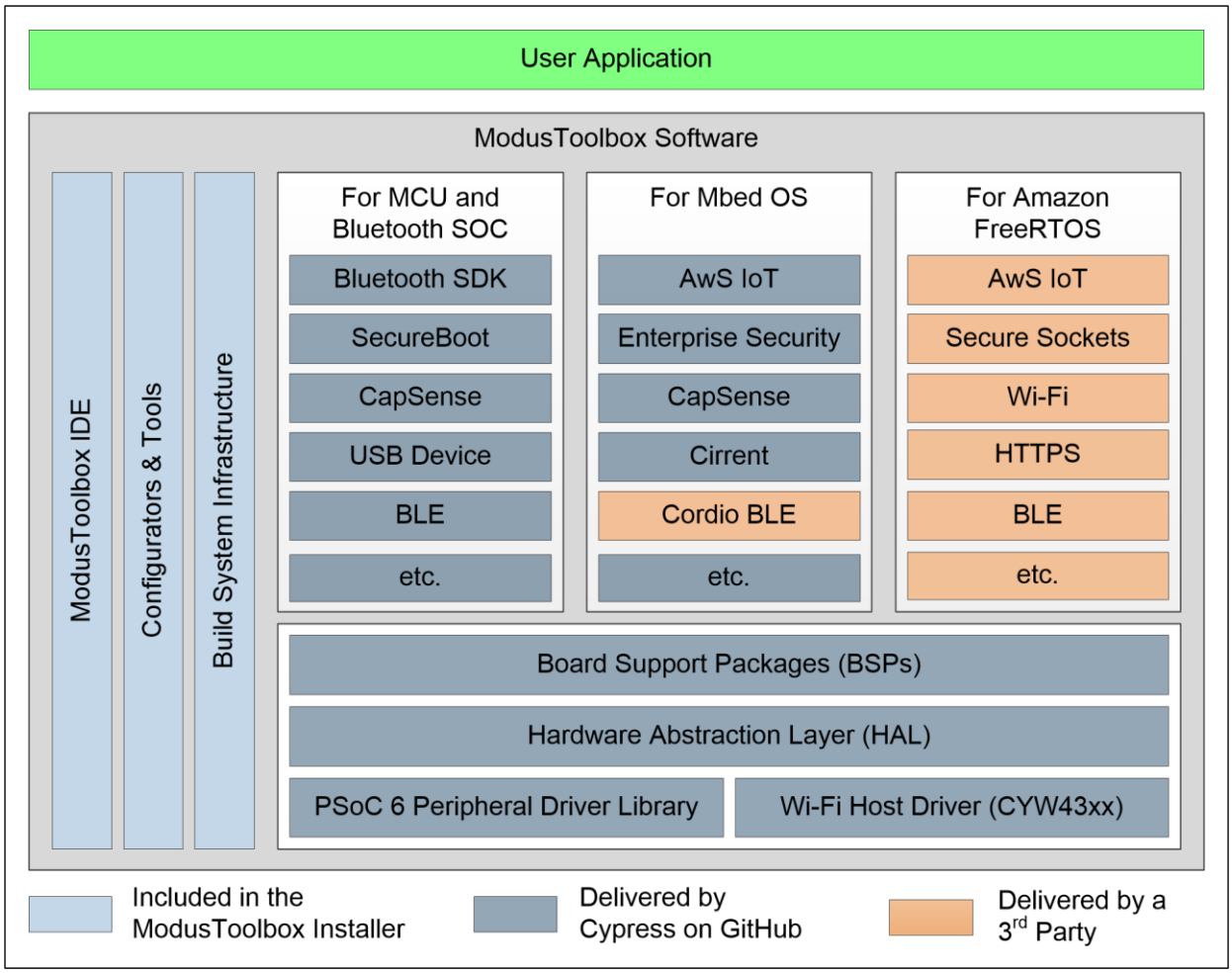
Some resources support all ecosystems. Others are specific to a particular ecosystem. The block diagram in Figure 1 is not a comprehensive list. However, it conveys the idea that, depending upon your programming domain, multiple resources are available to you. See Core Resources for available tools and Software Enablement Resources for available libraries and board support packages.

This document provides a high-level overview of ModusToolbox software v2.x, including:

- the ModusToolbox project creation and build tools
- available resources and how to get them
- which resources are supported for various targets and applications
- how to integrate the available resources into your project

This document does **not** discuss the details of any individual tool or library. Each resource has its own comprehensive documentation that covers the technical details.

Figure 1. Some ModusToolbox Resources



2. Core Resources

The ModusToolbox installer provides required and optional core resources for any project. This chapter provides an overview of the available resources:

- Build system infrastructure
- Configurators and tools
- Utilities
- ModusToolbox IDE

The installer does not include enablement software such as driver libraries or middleware.

Build System Infrastructure

The build system infrastructure is the fundamental resource in ModusToolbox software. It serves three primary purposes:

- create a project (either a folder that contains all the files required to build the software, or a ModusToolbox IDE project)
- create an executable
- provide debug capabilities

A makefile defines everything required for your project, including:

- the target hardware (board/board support package to use)
- the source code and libraries to use for the project
- the build tools to use
- compiler/assembler/linker flags to control the build

The build system automatically discovers all .c, .h, .cpp, .s, .a, .o files in the project folder and subfolders, and uses them in the project. The makefile can also discover files outside the project folder. You can add another directory using the `CY_SHAREDLIB_PATH` variable. You can also explicitly list files in the `SOURCES` and `INCLUDES` make variables.

Each library used in the project is identified by a *.lib* file. This file contains the URL to a git repository, and a commit tag. Cypress git repositories are on GitHub. For example, a capsense.lib file might contain the following line:

```
https://github.com/cypresssemiconductorco/capsense/#release-v2.0.0
```

The build system implements the `make getlibs` command. This command finds each *.lib* file, clones the specified repository, checks out the specified commit, and collects all the files in a single *libs* directory in the project folder. Typically the `make getlibs` command is invoked transparently when you create a project, although you can invoke the command directly from a command line interface. See Running ModusToolbox from the Command Line for detailed documentation on the build system infrastructure.

The simplest way to get started is to use an example application that targets the domain and ecosystem of interest, for example using CapSense in an Mbed OS application. Examples are available on the Cypress GitHub site. Whether you work from within the ModusToolbox IDE or from a command line, each invokes the same build system infrastructure.

ModusToolbox software supports two approaches to managing the files required for a project:

- Project Creator (no IDE required)
- New Application wizard (ModusToolbox IDE)

Either path invokes the makefile and build infrastructure transparently (`make getlibs` happens automatically). If you work from the command line, use `make getlibs` to create the project.

Creating a Project

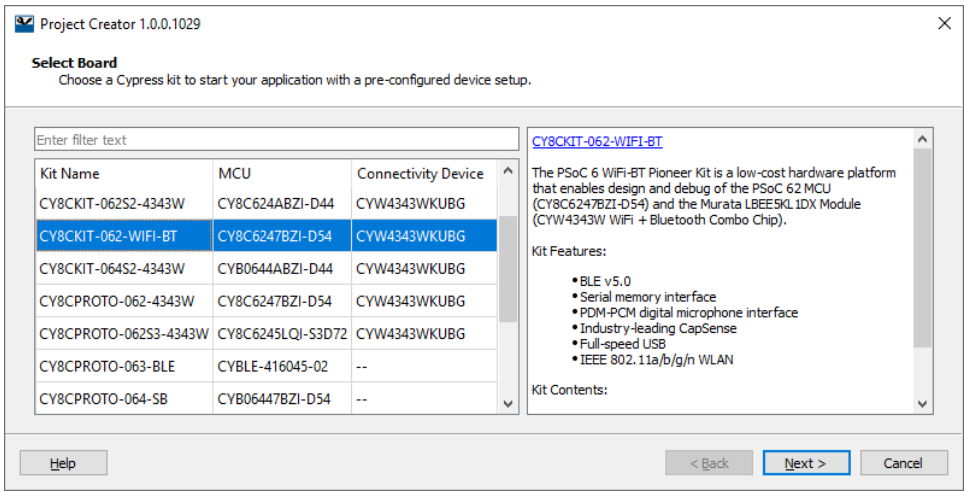
You can create a project using stand-alone tools, or use the same tools integrated within the ModusToolbox IDE. The process is essentially the same. You create a project, and then manage the libraries used in that project.

Project Creator (No IDE Required)

If your development process uses a third-party IDE, or you work from the command line, use the Project Creator to get started. The Project Creator is a simple wizard you use to select a kit and a starter application. See ModusToolbox Project Creator Guide for documentation.

First, launch the Project Creator tool from the *<ModusToolbox Install>/tools_x.y/project-creator/* folder. The screenshot shows the Windows-hosted tool. The same functionality is available on macOS and Linux. You select your board, select a starter application, and create the project.

Figure 2. Project Creator



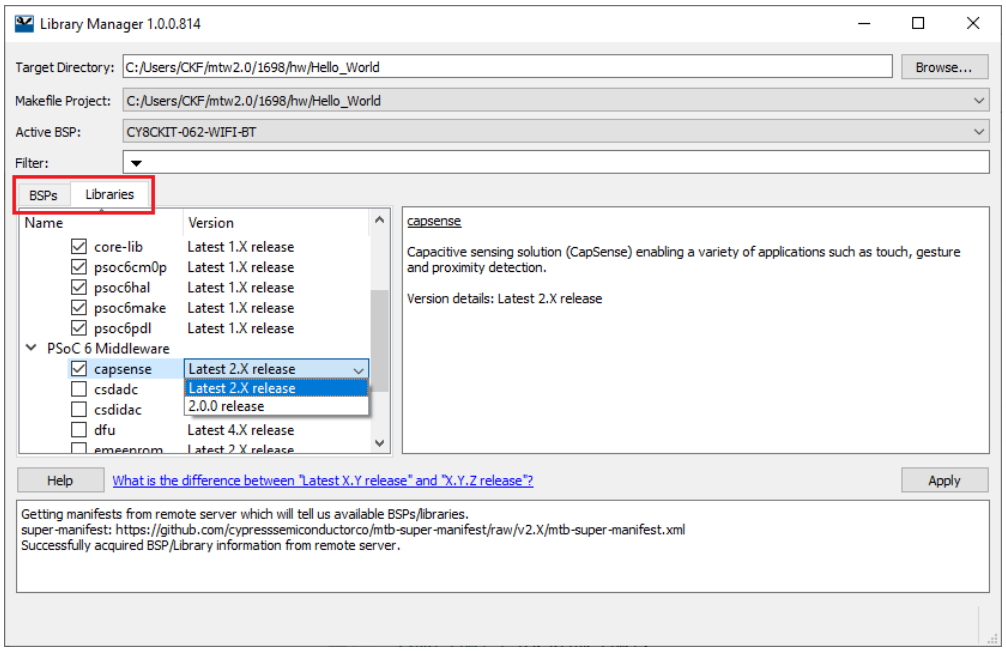
The Project Creator provides a local copy of each file in the project folder, so the application is completely self-contained.

There is one slight exception to this. Cypress provides many starter applications that use the Cypress Bluetooth SDK (BTSDK), for kits that support the BTSDK. To instantiate these applications, first create the *wiced_btsdk* application. This creates a local copy of the BTSDK. All Bluetooth starter applications reference that common SDK. The applications and the SDK must be in the same directory. The information in the Project Creator guides you for this case.

After the project is created, use the Library Manager tool to add or remove libraries, change to a different version of a library, or change the board used for the application (if the new board supports the application functionality). See the ModusToolbox Library Manager Guide documentation.

Launch the tool from the *ModusToolbox/tools_X.Y/library-manager/* folder. You specify the **Target Directory**, and the tool discovers all the makefiles within that tree. Select the **Makefile Project** you want to modify, and the **Active BSP** for use for that project appears. Click the current **Version** to see available choices.

Figure 3. Library Manager



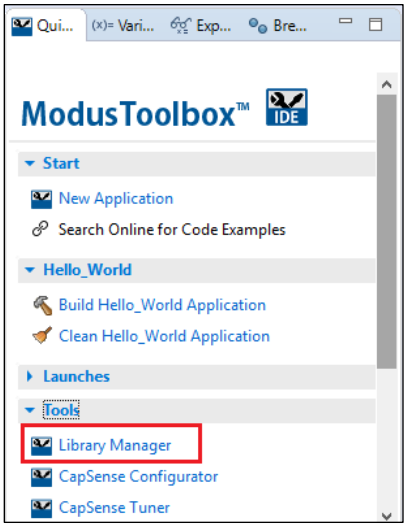
When you complete your changes and click **Apply**, the tool runs `make getlibs` to recreate the project using the BSP and libraries you have selected.

New Application Wizard (ModusToolbox IDE)

The IDE's New Application wizard is the same as the Project Creator tool, but also creates an IDE project. You select your board, select a starter application, and create the project. This gathers all the files, puts them in a project folder in the Eclipse workspace, and creates a project in the IDE. The New Application wizard also creates Eclipse-specific launch configurations for debugging.

After the project is created, use the Library Manager to add or remove libraries, change to a different version of a library, or change the board used for the application (if the new board supports the application functionality). In the ModusToolbox IDE, you select your project in the Project Explorer, then click the **Library Manager** link in the Quick Panel.

Figure 4. Launching the Library Manager



Creating an Executable

The build infrastructure provides several variables you use to control the build. There are also variables you can use to pass compiler and linker flags to the tool chain.

Table 1. Basic `make` Variables

Variable	Description
TARGET	Specifies the target board/kit. For example, CY8CPROTO-062-4343W
APPNAME	Specifies the name of the application
TOOLCHAIN	Specifies the build tools used to build the application
CONFIG	Specifies the configuration option for the build [Debug Release]
VERBOSE	Specifies whether the build is silent or verbose [true false]

ModusToolbox software is tested on these tool chains:

Table 2. Supported Tool Chains

Variable value	Tools	Host OS
`GCC_ARM`	GNU Arm Embedded Compiler v7	Mac OS, Windows, Linux
`ARM`	Arm compiler v6	Windows, Linux
`IAR`	Embedded Workbench v8.2	Windows

In the makefile, set the TOOLCHAIN variable to the build tools of your choice. For example: `TOOLCHAIN=GCC_ARM`.

ModusToolbox software installs the GNU Arm tool chain and uses it by default. If you wish to use another tool chain, you must provide it and specify the path to the tools. For example, `CY_COMPILER_PATH=<yourpath>`. If this path is blank, the build infrastructure looks in the ModusToolbox install folder.

Whether from within the ModusToolbox IDE or the command line, the same build system compiles and links the code. In the ModusToolbox IDE click the **Build Application** link in the Quick Panel. From the command line, use `make build`.

Because the IDE relies on the build infrastructure, it does not use the standard Eclipse GUI to modify build settings. It uses the build options specified in the makefile. This design ensures that the behavior of the project, its options, and the `make` process is

consistent regardless of the development environment and workflow. See Running ModusToolbox from the Command Line for detailed documentation on the build system infrastructure.

If you work in a different IDE, you manage the build using the features and capabilities of that IDE.

Program and Debug Support

ModusToolbox software supports the Open On-Chip Debugger (OpenOCD) using a GDB server, and supports the J-Link debug probe. For the Mbed OS ecosystem, ModusToolbox supports Arm Mbed DAPLink.

The ModusToolbox IDE can program devices and establish a debug session. For programming, Cypress Programmer is available separately. It is a cross-platform application for programming Cypress PSoC 6 devices. It can program, erase, verify, and read the flash of the target device.

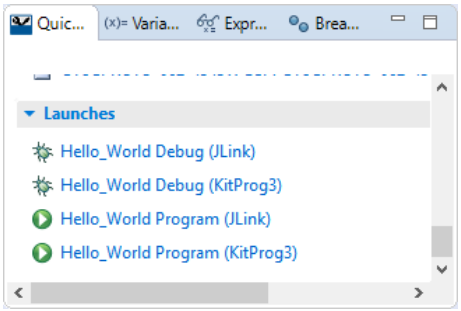
Cypress Programmer and ModusToolbox IDE use KitProg3 low-level communication firmware. The firmware loader (fw-loader) is a software tool you can use to easily switch back and forth between KitProg2 and KitProg3, if you need to do so. The fw-loader tool is installed with the ModusToolbox software. It is also available separately in a GitHub repository.

Table 3. ModusToolbox Program & Debug Tools

Tool	Description	Documentation
Cypress Programmer	Cypress Programmer functionality is built into ModusToolbox Software. Cypress Programmer is also available as a standalone tool.	Programming Tools page, go to the documentation tab
fw-loader	A simple command line tool to identify which version of KitProg is on a Cypress kit, and easily switch back and forth between legacy KitProg2 and current KitProg3.	*readme.txt* file in the tool folder
KitProg3	This tool is managed by fw-loader, it is not available separately. KitProg3 is Cypress' low-level communication/debug firmware that supports CMSIS-DAP and DAPLink (for Mbed OS). Use fw-loader to upgrade your kit to KitProg3, if it has KitProg2 installed.	User Guide
OpenOCD	A Cypress-specific implementation of OpenOCD is installed with ModusToolbox software.	Developer's Guide
DAPLink	Support is implemented through KitProg3	DAPLink Handbook

In the ModusToolbox IDE, the application creation process generates launch configurations for the Eclipse IDE. Some configurations appear in the Quick Panel. Use **Run** > **Debug Configurations** to see all available configurations.

Figure 5. Quick Panel Launch Configurations



See the ModusToolbox User Guide for documentation on debug support from within the IDE.

Configurators and Tools

While it is possible to write configuration code from scratch, the effort to do so is considerable. ModusToolbox software includes configurators to make it easier to configure a hardware block or a middleware library. For example, instead of having to search

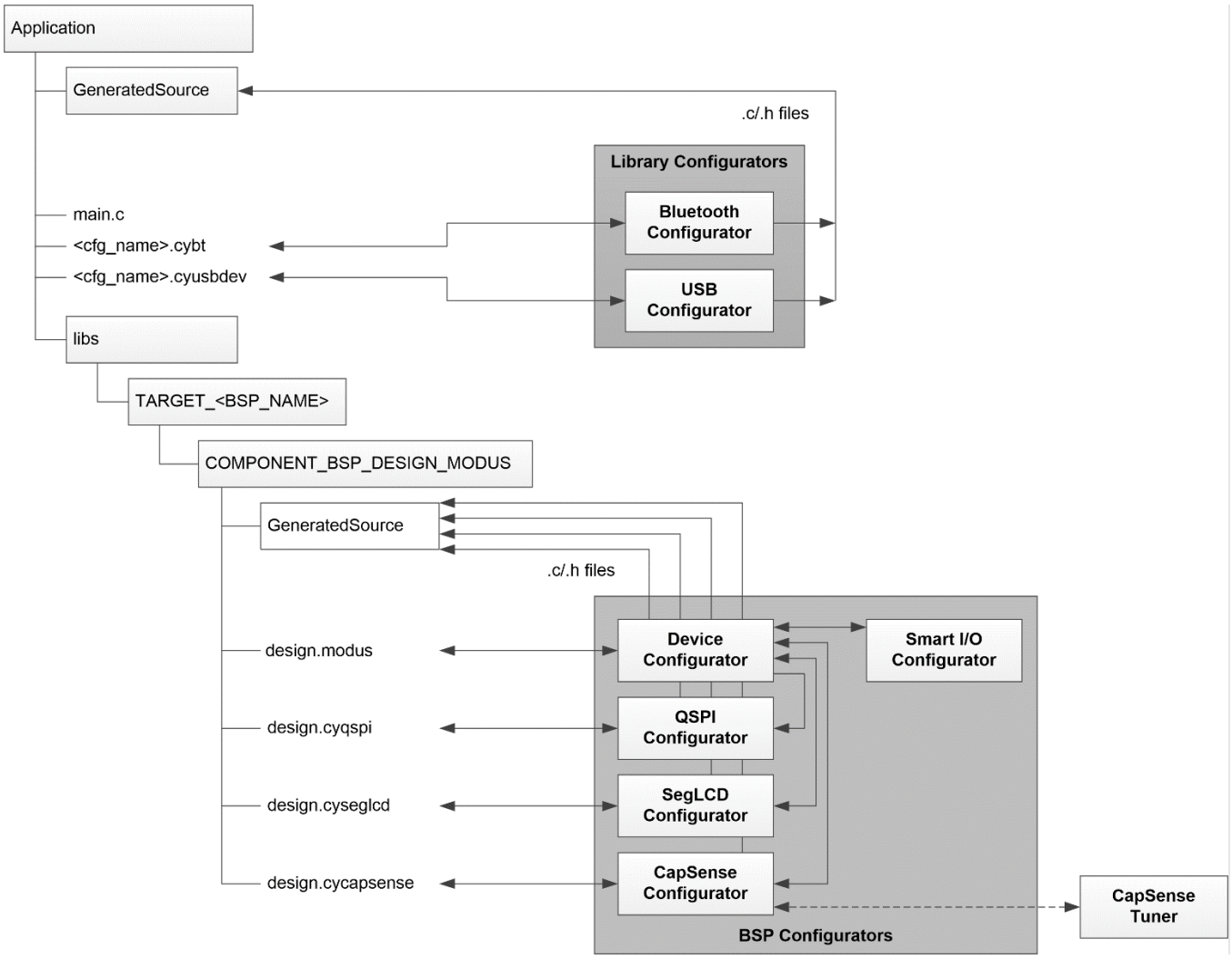
through all the documentation for information on how to set up a serial communication block as a UART with a desired configuration, open the appropriate configurator to set the baud rate, parity, stop bits, etc.

Each configurator is a cross-platform tool that allows you to set configuration options for the corresponding hardware peripheral or library. When you save a configuration, the tool generates the C code or configuration file used to initialize the hardware or library with the desired configuration.

Configurators are independent of each other, but are typically used together to provide complete configuration options. They can be used stand alone, or launched from within the ModusToolbox IDE. The typical workflow is to start with the Device Configurator. As you enable features that are managed by other configurators, links to those configurators are available. You can also launch configurators independently.

The following diagram shows one example of how configurators might be used in a typical application:

Figure 6. Configurators and Generated Code



BSP Configurators configure the hardware on a specific device. This can be a board provided by Cypress, a Cypress partner, or a board that you create that is specific to your application. Some of these configurators interact with the *design.modus* file to store and communicate configuration settings between different configurators. Code generated by a BSP Configurator is stored in a directory named *GeneratedSource*, which is in the same directory as the *design.modus* file. This is generally located in the BSP for a given target board.

Library configurators support configuring application middleware. Library configurators do not read nor depend on the *design.modus* file. They generally create data structures to be consumed by software libraries. These data structures are specific to the software library and independent of the hardware. Configuration data is stored in a configurator-specific XML file (for example, *.cybt, *.cyusbdev). Any source code generated by the configurator is stored in a *GeneratedSource* directory in the same directory as the XML file.

Note While the configuration file is generally stored in the application directory, this is not a requirement.

ModusToolbox software includes other tools that provide support for project creation, device firmware updates, and so on. All tools are installed by the ModusToobox Installer. With rare exception each tool has a user guide located in the *docs* folder beside the tool itself. Most user guides are also available online.

Table 4. ModusToolbox Configurators & Tools

BSP Configurator	**Details**	**Documentation**
device-configurator	Configure device peripherals, clocks, pins, and DMA. Launch other configurators independently or from within the Device Configurator. Generates code based on your choices.	User Guide
capsense-configurator	Create and configure CapSense widgets, and generate code to control the application firmware.	User Guide
capsense-tuner	Tune the performance and sensitivity of CapSense widgets.	User Guide
qspi-configurator	The Quad Serial Peripheral Interface generates the code to configure external memory on your hardware.	User Guide
smartio-configurator	Configure Smart I/O™, adds programmable logic to an I/O port.	User Guide
seglcd-configurator	Configure a generic LCD Direct Segment Drive controller for a variety of LCD glass at different voltage levels with multiplex ratios up to 16x.	User Guide
Library Configurator	**Details**	**Documentation**
bt-configurator	Generate configuration code for Bluetooth applications, including the Generic Attribute Profile (GATT) database, Service Discovery Protocol (SDP) database, Generic Access Profile (GAP) configuration, Logical Link Control and Adaption Protocol (L2CAP), and Link Layer parameters.	User Guide
usbdev-configurator	Define USB descriptors and generate configuration, header, and source files used by the USBDev middleware.	User Guide
Other Tools	**Details**	**Documentation**
project-creator	Create a new application project. This tool is a stand-alone version of the IDE wizard, available as a GUI and a command-line tool (CLI).	User Guide
library-manager	Add, remove, or update libraries and BSP used in an application; edits the makefile	User Guide
fw-loader	Update KitProg communication firmware on a kit. Also available separately on GitHub	*readme.txt* file in the tool folder
cymcuelftool	Merges CM0+ and CM4 application images into a single executable. Typically launched from a post-build script. This tool is not used by most applications.	User Guide is in the tool's *docs* folder
dfuh-tool	Use the Device Firmware Update Host tool to communicate with a PSoC® 6 MCU that has already been programmed with an application that includes device firmware update capability. Provided as a GUI and a command-line tool. Depending on the ecosystem you target, there may be other over-the-air firmware update tools available.	User Guide
cype-tool	The power estimator tool provides an estimate of the power consumed by a target device.	User Guide

Utilities

ModusToolbox software includes some additional utilities that are often necessary for application development. In general you use these utilities transparently.

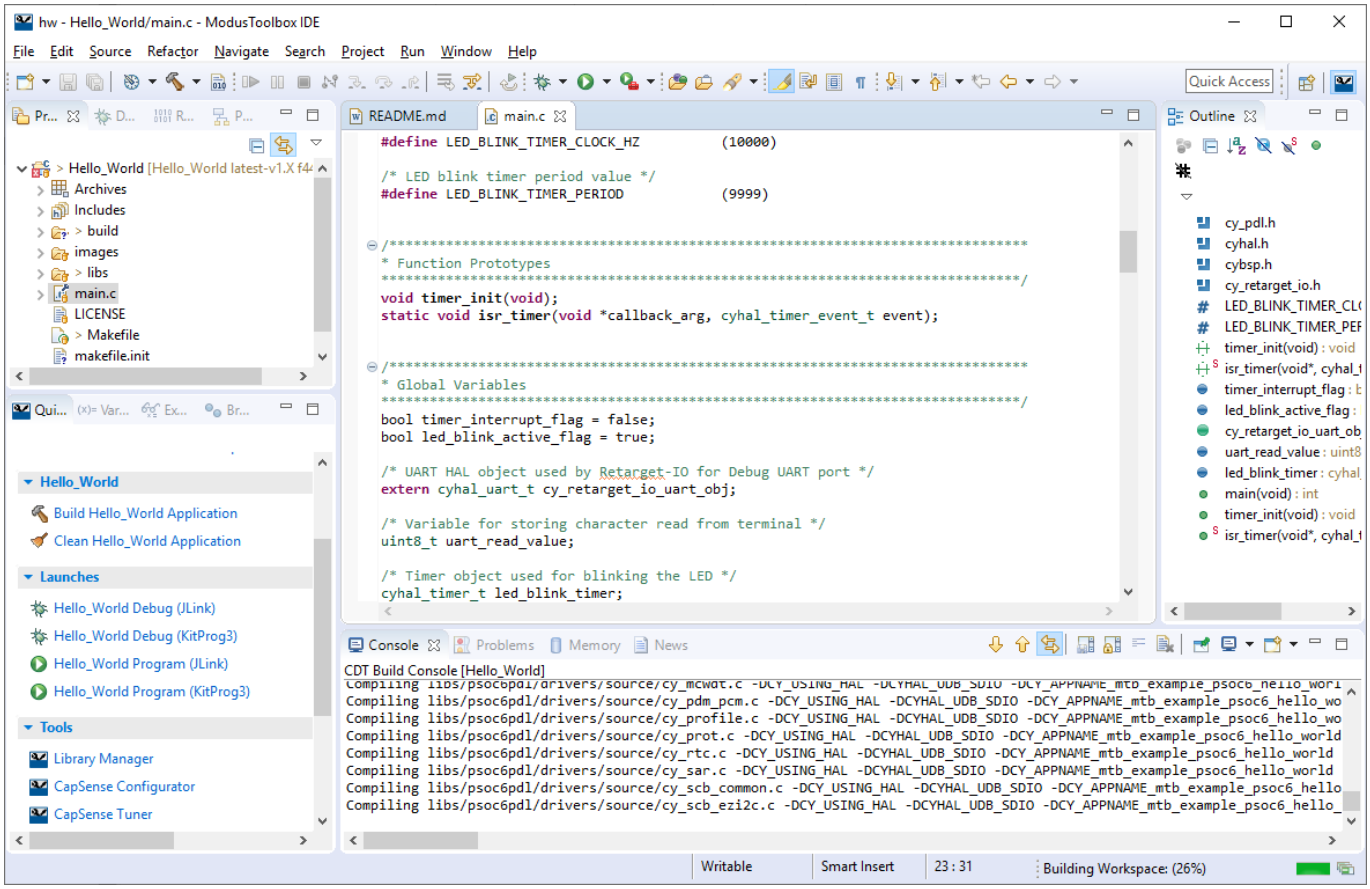
Table 5. ModusToolbox Utilities

Utility	Description
GCC	Supported toolchain installed by ModusToolbox.
GDB	The GNU Project Debugger is installed as part of GCC.
OpenOCD	The Open On-Chip Debugger provides a debugging and programming interface for embedded systems.
JRE	Java Runtime Environment; required by various applications and backend processes.

ModusToolbox IDE

ModusToolbox IDE is a full-featured, cross-platform, Eclipse-based IDE. It includes project management, code authoring and editing, build tools, and debug capabilities. ModusToolbox IDE supports the C and C++ programming languages. It includes the GCC Arm build tools. It supports debugging via OpenOCD or J-Link. You can use the IDE to develop applications using ModusToolbox software. The IDE is optional.

Figure 7. ModusToolbox IDE



Refer to the ModusToolbox IDE User Guide for documentation on the IDE.

3. Software Enablement Resources

This chapter organizes enablement software in these broad resource categories:

- Low-level resources
- Board support packages (BSPs)
- Middleware
- Code examples

ModusToolbox software targets three primary software development flows:

- PSoC 6 MCU and Bluetooth SoC ecosystem development
- Mbed OS ecosystem development
- Amazon FreeRTOS ecosystem development

As discussed in Build System Infrastructure, to include a resource a starter application specifies a.*lib* file, which provides the URL and commit for the required code. The build system copies the files into your project directory. This means that if you use the Project Creator or New Application wizard, all required files appear automatically. However, each software enablement resource from Cypress is available separately in a GitHub repository that typically includes both source code and documentation.

BSPs typically include one or more resources automatically. For example, any BSP that targets a PSoC 6 device includes the PDL driver library automatically. If the board supports CapSense, the BSP includes the CapSense library.

At a higher level, Cypress starter applications (code examples) include the BSP for the supported kit, along with any other required middleware. As a result, the libraries that the example depends upon are brought into the project automatically.

Because the libraries are freely available on GitHub, you may download any library to create a local copy. You can then refer to the library from multiple projects in your development environment, should you prefer.

Some significant resources are available as part of a supported ecosystem and not provided by Cypress. For example, the Mbed Transport Layer Security (TLS) library or the Cordio Bluetooth stack are part of the Mbed ecosystem. You include ecosystem-specific resources using whatever mechanism is defined in that ecosystem.

Low-Level Resources

Low-level resources are related to specific device features. For example, a low-level driver (LLD) contains the API and source code to configure and use a feature or peripheral on a device. ModusToolbox provides the Peripheral Driver Library (PDL) for PSoC 6 devices, or the Wi-Fi Host Driver (WHD) for CYW43xx connectivity devices. Device-specific source code and header files are included in the LLD.

In addition, Cypress provides a hardware abstraction layer (HAL). You can use the HAL for most hardware configuration. It abstracts some of the complexities of using a low level driver directly. For example, a BSP typically uses HAL functions to configure the hardware. In cases where your application requires driver features not supported in the HAL, you can use driver library function calls directly. The HAL and the driver libraries are compatible.

ModusToolbox configurators also generate code (particularly configuration structures) that you can use to configure hardware. In general, the configurators work with the PSoC 6 PDL directly and do not use the HAL. If you use a configurator to configure a peripheral, the HAL will not modify that configuration.

This design means that you can mix and match HAL function calls, direct driver library function calls, and configurator generated source.

The abstraction-rtos library provides a common API that retargets your call to the appropriate RTOS-specific function. This currently supports the FreeRTOS and RTX kernels. Should you wish to use the abstraction-rtos library with a different RTOS, you can examine the API and redirect calls to your RTOS.

Note that ModusToolbox configurators generate PDL-specific configuration structures and function calls. That code requires the PDL to be part of the application project. BSPs always include the PDL when necessary.

Table 6. Low-level Resources

Item	Details	Docs	MCU & BT SOC	Mbed OS	Amazon FreeRTOS
psoc6hal	The Hardware Abstraction Layer (HAL) provides a high-level interface to configure and use hardware blocks on Cypress MCUs. It is a generic interface that can be used across multiple product families. The focus on ease-of-use and portability means the HAL does not expose all of the low-level peripheral functionality	API Reference	✓	✓	✓
abstraction-rtos	A common API that allows code or middleware to use RTOS features without knowing what the RTOS is	API Reference	✓	✓	✓
core-lib	Provides header files that declare basic types and utilities (such as result types or ASSERT) that can be used by multiple BSPs	API Reference	✓	✓	✓
retarget-io	Provides a board-independent API to retarget text input/ouput to a serial UART on a kit	API Reference	✓	✓	✓
rgb-led	Provides a board-independent API to use the RGB LED on a kit	API Reference	✓	✓	✓
serial-flash	Provides a board-independent API to use the serial flash on a kit	API Reference	✓	✓	✓
psoc6pdl	Peripheral driver library for PSoC 6 devices. The library is device-independent, so can be precompiled and used for any PSoC 6 MCU device or project. Included automatically by any BSP targeting a PSoC 6 device	API Reference	✓	✓	✓
wifi-host-driver	Driver library for Cypress WLAN devices (CYW43xxx) that can be easily ported to popular RTOSs such as Amazon FreeRTOS and Mbed OS. The wifi-host-driver is included automatically by board support packages that require this driver (those with CYW43xx devices)	API Reference	✓	✓	✓
psoc6cm0p	Prebuilt application images for the Cortex M0+ CPU of the dual-CPU PSoC 6 devices. The images are provided as C arrays ready to be compiled as part of the Cortex M4 application. The Cortex M0+ application code is placed to internal flash by the Cortex M4 linker script.	See the readme file in the repository	✓	✓	✓
psoc6make	This repository provides the build recipe makefiles and scripts for building and programming PSoC 6 applications. You can build an application either through a command-line interface (CLI), the ModusToolbox IDE, or a third-party IDE.	See the readme file in the repository	✓	✓	✓
Mbed OS HAL	Mbed's hardware abstraction layer. Support for Cypress targets is available from the Mbed OS archive. This HAL is part of the Mbed ecosystem and not provided directly by Cypress.	Mbed API documentation		✓	

Board Support Packages and Kits

BSPs are aligned with Cypress kits; they provide files for basic device functionality. A BSP typically has a *design.modus* file that configures clocks and other board-specific capabilities. That file is used by the ModusToolbox configurators. (See Configurators and Tools). A BSP also includes the required device support code for the device on the board. Users can modify the configuration to suit their application. A BSP uses low-level resources to add functionality. For example, a BSP typically adds the following libraries, as appropriate for the kit/device:

- core-lib – to implement return types and generic functionality useful for any kit
- psoc6hal – to implement the ModusToolbox hardware abstraction layer
- psoc6cm0p – to add a predefined CM0+ application
- psoc6make – to implement the build system infrastructure
- capsense – if appropriate for the kit

Application-specific functionality is added by the starter application makefile. For example, if the example uses the RGB LED on a kit, it will include the rgb-led library in its makefile.

Cypress releases BSPs independently of ModusToolbox software as a whole. This search link finds all currently available BSPs on the Cypress GitHub site.

The search results include links to each repository, named TARGET_<kit number>. For example, you will find links to repositories like TARGET_CY8CPROTO-062-4343W. Each repository provides links to relevant documentation. The following links use this BSP as an example. Each BSP has its own documentation.

The information provided varies, but typically includes one or more of:

- an API reference for the BSP
- the general BSP User Guide
- a link to the associated kit page with kit-specific documentation

A BSP is specific to a board and the device on that board. For custom development, you can create or modify a BSP for your device. See the BSP User Guide for how BSPs work and how to create your own for a custom board.

Middleware

This category includes any library that implements an API for a particular domain, for example capacitive sensing or an http server. A middleware library may be created by Cypress or come from a third party. Cypress-created middleware may use the Cypress HAL or an LLD directly. In that case, you need the corresponding driver library or BSP for the middleware to work. Any example application that requires a library downloads that library automatically.

The Amazon FreeRTOS ecosystem provides a collection of libraries that provide significant connectivity and other capabilities beyond the FreeRTOS kernel and its internal libraries. This includes interaction with AWS IoT services. Those libraries are not listed here.

Table 7. Middleware Resources

Connectivity Middleware	Description	Docs	MCU & BT SOC	Mbed OS	Amazon FreeRTOS
btsdk-audio	The SDK features a dual-mode Bluetooth stack with stack- and profile-level APIs for embedded BT application development. It supports GAP, GATT, SMP, RFCOMM, SDP, AVDT/AVCT and BLE Mesh protocols, as well as over-the-air upgrade. The Bluetooth SDK is factored into a collection of smaller libraries, so that you can download and use those parts of the SDK necessary for your application.	btsdk-docs	✓		
btsdk-ble					
btsdk_drivers					
btsdk-hid					
btsdk-include					
btsdk-mesh					
btsdk-ota					
btsdk-rfcomm					

btsdk-tools btsdk-utils btsdk-host-apps-bt-ble btsdk-host-apps-mesh btsdk-peer-apps-ble btsdk-peer-apps-mesh btsdk-peer-apps-ota	Follow instructions in the ModusToolbox IDE new application wizard or the stand-alone Project Creator tool to create a local copy of the entire Bluetooth SDK.				
aws-iot	Provides secure, bi-directional communication between Internet-connected devices such as sensors, actuators, embedded micro-controllers, or smart appliances and the AWS Cloud. Supports MQTT and HTTP protocols.	Developer Guide		✓	Available as part of Amazon FreeRTOS
enterprise-security	This library implements a collection of the most commonly used Extensible Authentication Protocols (EAP) used in enterprise WiFi networks	API Reference		✓	
http-server	Provides communication functions for an HTTP server.	GitHub readme		✓	
connectivity-utilities	General purpose middleware connectivity utilities, for instance a linked_list or a json_parser	See the code for each		✓	
bless	The Bluetooth Low Energy Subsystem (bless) library contains a comprehensive API to configure the BLE Stack and the underlying chip hardware. It incorporates a Bluetooth Core Specification v5.0 compliant protocol stack. You may access the GAP, GATT and L2CAP layers of the stack using the API.	API Reference	✓		
Arm Mbed Cordio	An open source Bluetooth Low Energy (BLE) solution offering both host and controller subsystems, with abstraction interfaces for both RTOS and hardware. It is part of the Mbed OS ecosystem and not provided by Cypress.	Mbed OS documentation		✓	
PSoC 6 Middleware	**Description**	**Docs**	**MCU**	**Mbed OS**	**Amazon FreeRTOS**
capsense	Cypress capacitive sensing solution. Capacitive sensing can be used in a variety of applications and products where conventional mechanical buttons can be replaced with sleek human interfaces to transform the way users interact with electronic systems.	API Reference	✓	✓	
csdadc	Enables the ADC or IDAC functionality of the CapSense Sigma-Delta hardware block. Useful for devices that do not include other ADC/IDAC options. The CSD HW block enables multiple sensing capabilities on PSoC devices including self-cap and mutual-cap capacitive touch sensing solutions, a 10-bit ADC, IDAC, and Comparator.	API Reference	✓	✓	
csdidac		API Reference	✓	✓	
usbdev	The USB Device library provides a full-speed USB 2.0 Chapter 9 specification compliant device framework. It uses the USBFS driver from PDL. The middleware supports Audio, CDC, and HID, and other classes. Use the USB Configurator tool to construct the USB Device descriptor	API Reference	✓	✓	
dfu	The Device Firmware Update (DFU) library provides an API for updating firmware images. You can create an application loader to receive and switch to	API Reference	✓	✓	

	the new application, and a loadable application to be transferred and programmed.				
Secure Boot SDK	This SDK includes all required libraries, tools, and sample code to provision and develop applications for PSoC 64 MCUs.	User Guide	✓	✓	
emeeprom	The Emulated EEPROM library provides an API to manage an emulated EEPROM in flash. It has support for wear leveling and restoring corrupted data from a redundant copy.	API Reference	✓	✓	
Other Middleware	**Description**	**Docs**	**MCU**	**Mbed OS**	**Amazon FreeRTOS**
freertos	FreeRTOS kernel, distributed as standard C source files with configuration header file, for use with the PSoC 6 MCU.	**FreeRTOS web page**	✓	✓	
emwin	Segger embedded graphic library and graphical user interface (GUI) framework designed to provide processor- and display controller-independent GUI for any application that needs a graphical display.	Overview	✓	✓	
Arm Mbed TLS	A library to include cryptographic and SSL/TLS capabilities in an embedded application. It is part of the Mbed OS ecosystem and not provided by Cypress.	API Reference		✓	

Code Examples

All current ModusToolbox examples can be found through the GitHub code example page. There you will find links to examples for the Bluetooth SDK, AWS IoT, Mbed OS, and PSoC 6 MCU, among others.

ModusToolbox code examples are example applications. In the ModusToolbox build infrastructure any example application that requires a library downloads that library automatically. Follow the directions in the code example repository to instantiate the example. Instructions vary based on the nature of the application and the targeted ecosystem.

Revision History

Document Title: ModusToolbox™ Software Overview Document Number: 002-28029		
Revision	**Date**	**Description of Change**
**	10/18/2019	New document.
*A	10/23/2019	Update Table 7, middleware resources; fix some links; rewrite the BSP section