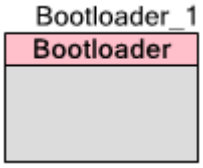
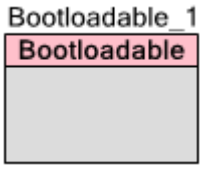
Bootloader and Bootloadable

1.30





Features

- Separate Bootloader and Bootloadable components
- Configurable set of supported commands
- Flexible component configuration

General Description

The bootloader system manages the process of updating the device flash memory with new application code and/or data. To make the process work we use these components:

- Bootloader project: project with Bootloader and Communication components
- Bootloadable project: project with a Bootloadable component, which creates the code

Bootloader Component

The Bootloader component allows you to update the device flash memory with new code. The bootloader accepts and executes commands, then passes command responses back to the communications component. The bootloader collects and arranges the received data and manages the actual writing of flash through a simple command/status register interface.

The project application type needs to match the component placed on the schematic. As an example for a bootloader project, set the **Application Type** to Bootloader (under Build Settings) and place a Bootloader component onto the schematic. For information about application types, see the PSoC Creator Help.

The bootloader manages the communications protocol to receive commands from an external system and pass those commands to the bootloader. It also passes command responses from the bootloader back to the off-chip system.

Architecture	Supported Interfaces				
	Custom Interface	USB	UART	I²C	SPI
PSoC 3 / PSoC 5LP	✓	✓	✓	✓	✓
PSoC 4	✓			✓	

Notes:

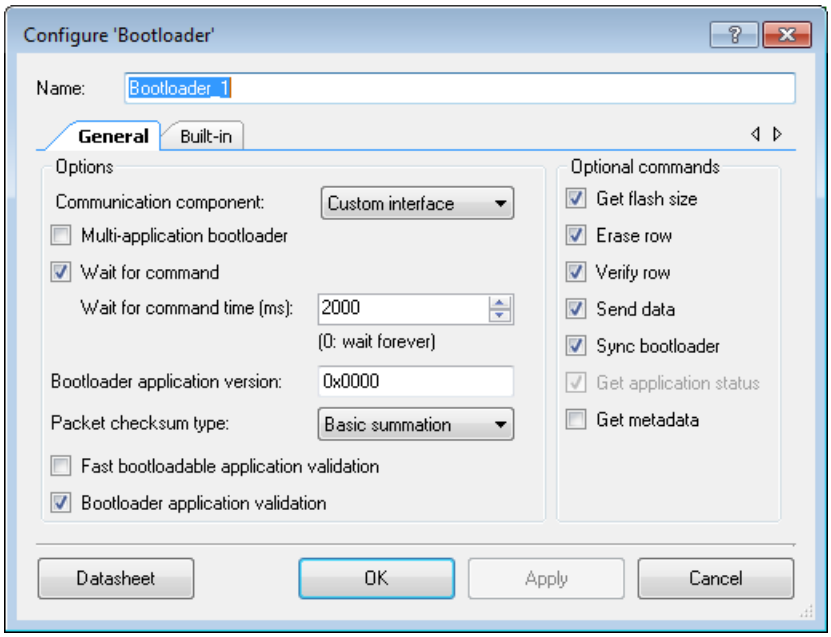
- The I^2C interface on PSoC 4 is implemented with the SCB component.
- The Custom Interface option allows adding bootloader support to any existing communications component. See the corresponding communications component datasheet for more details about the appropriate communication method.
- For PSoC 4000 devices, each update to a flash row will automatically modify the clock settings for the device. Writing to flash requires that changes be made to the IMO and HFCLK settings. The configuration is restored after each row is written.
  - HFCLK will have several frequency changes during each write to a flash row between a minimum frequency of the current IMO frequency divided by 8 and a maximum frequency of 12 MHz.
  - These clock changes will impact the operation of the communications component and any other hardware that is present in the bootloader project.
  - The I^2C slave component is tolerant of clock changes, but the clock changes can result in a NAK response when transactions occur during a row write. The bootloader host should be designed to retry in this case.

Bootloadable Component

When you use the Bootloadable component, you can specify additional parameters for the bootloadable project.

Bootloader Component Parameters

Drag a Bootloader component onto your design and double-click it to open the **Configure** dialog.



The Bootloader component has the following parameters:

Communication component

This is the communications component that the bootloader uses to receive commands and send responses. Select one, and only one, communications component. This property is a list of the available communications protocols on the schematic that have bootloader support. In all cases, independent of what is on the schematic, there is also a custom interface option available that allows for implementing the bootloader functions directly. For information and instructions on how to do this, see the *Component Author Guide*.

If there is no communications component on the schematic, then the Custom Interface option is selected. This allows for implementing the communication in any way. See the corresponding component datasheet for more details about the appropriate communication method.

Multi-application bootloader

This option allows two bootloadable applications to reside in flash. It is useful for designs that require a guarantee that there is always a valid application that can be run. This guarantee comes with the limitation that each application has one half of the flash available from what would have been available for a "standard" bootloader project.

Wait for command

On device reset, the bootloader can wait for a command from the bootloader host or jump to the application code immediately. If this option is enabled, the bootloader waits for a command from the host until the timeout period specified by **Wait for command time** parameter occurs. If the bootloader does not receive this command within the specified timeout interval, the active bootloadable project in the flash is executed after the timeout.

Wait for command time

If the bootloader waits for the command to start loading a new bootloadable application after a reset, this is the amount of time it waits before starting the existing bootloadable application. This option is valid only if **Wait for command** is enabled, otherwise it is ignored and grayed out. The zero value is interpreted as wait forever. The default value is a 2 second time out.

Bootloader application version

This parameter provides a 2 byte number to represent the version of the Bootloader application. Default value is 0x0000.

Packet checksum type

This parameter has a couple of options for the type of checksum to use when transferring packets of data between the host and the bootloader. The default value is **Basic summation**.

The basic summation checksum is computed by adding all the bytes (excluding the checksum) and then taking the 2's complement. The other option is CRC-16CCITT – the 16 bit CRC using the CCITT algorithm.

The checksum is computed for the entire packet with the exception of the Checksum and End of Packet fields.

Fast bootloadable application validation

This option controls how the bootloader verifies the application data. If it is disabled, the bootloader computes the bootloadable application checksum every time before starting it. If enabled, the bootloader only computes the checksum the first time and assumes that it remains valid in each future startup.

Bootloader application validation

If this option is enabled, the bootloader validates itself by calculating the checksum and comparing it with the saved one that resides in metadata. If the validation is not passed, the device is halted. If this option is disabled, the bootloader is executed even if it is corrupted. This could lead to unpredictable results.

Optional Commands

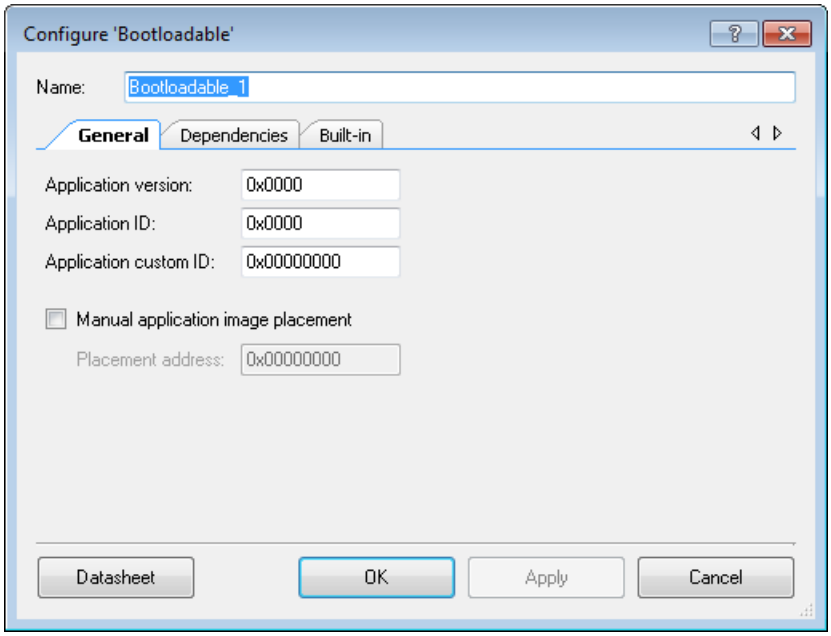
This group of options determines whether or not a corresponding command is supported by the bootloader. If it is enabled, then the corresponding command is supported. By default all optional commands are supported.

The **Get flash size**, **Send data**, and **Verify row** commands are required by the Cypress Bootloader Host tool. These commands might not be used by custom bootloader host tools.

Bootloadable Component Parameters

Drag a bootloadable component onto your design and double-click it to open the **Configure** dialog.

General Tab



The **General** tab of the Bootloadable component contains the following parameters:

Application version

This parameter provides a 2 byte number to represent the version of the bootloadable application. Default value is 0x0000.

Application ID

This parameter provides a 2 byte number to represent the ID of the bootloadable application. The default value is 0x0000.

Application custom ID

This parameter provides a 4 byte custom ID number to represent anything in the bootloadable application. The default value is 0x00000000.

Manual application image placement

If this option is enabled, PSoC Creator places the bootloadable application image(s) at the location specified by **Placement address** option. It is also placed according to the rules outlined in section **Bootloadable Project** below.

Use this option independently for each of two bootloadable applications, if both of them are referenced to the **Multiapplication bootloader** application.

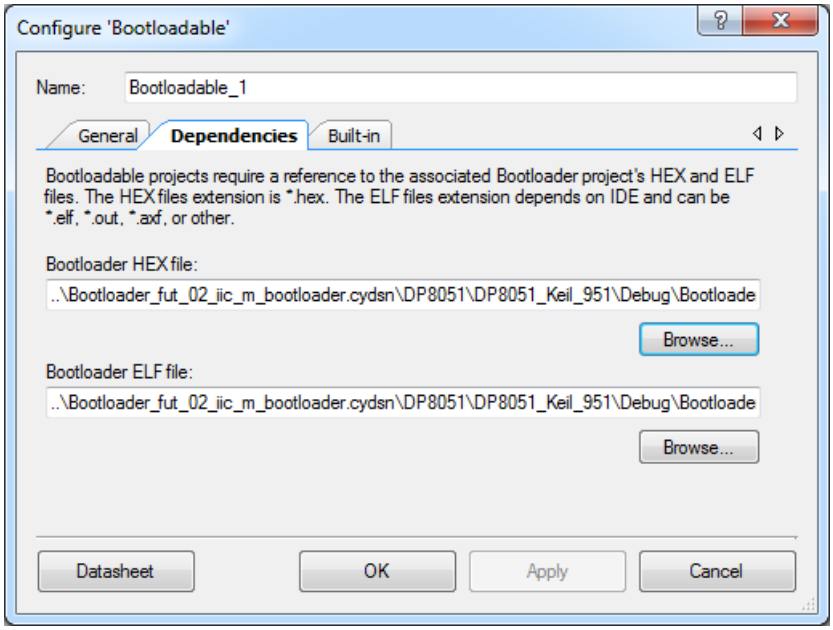
Placement Address

This option allows you to specify the address where the bootloadable application is placed in memory. This option is only valid if you enable the **Manual application image placement** option; otherwise it is grayed out. You need to specify the address above the bootloader image and below the metadata area.

You calculate the placement address by multiplying the number of the flash row (starting from which the image is placed) by the flash row size and adding result to the flash base address. Align the placement address to the flash row size. See the *Flash and EEPROM* chapter of the *System Reference Guide* for details about flash memory organization.

You get the first available row for the bootloadable application from the associated cyacd file when the **Manual application image placement** option is disabled or can be reported by the Get Flash Size command.

Dependencies Tab



The **Dependencies** tab of the Bootloadable component contains the following parameters:

Bootloader HEX file

This option allows you to associate a bootloadable project with the bootloader project HEX file. This is necessary so that the build of the bootloadable project gets the information about the bootloader project (for example, properly calculate where it belongs in memory).

Bootloader ELF file

This option allows you to associate a bootloadable project with the bootloader project ELF file. The ELF file extension depends on IDE. For example, PSoC Creator generates ELF files with *.elf extension, while other IDEs produce *.elf, *.out, or *.axf files.

This option is automatically populated with the path to the *.elf file, if it is located in the same folder with the specified HEX file. You can always update this option and specify the path to the ELF file manually.

Note Make sure that HEX and ELF files are generated by the same build process to ensure that they are coherent.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. This table lists and describes the interface to each function. The following sections cover each function in more detail.

By default, PSoC Creator assigns the instance name “Bootloader_1” to the first instance of a Bootloader component and “Bootloadable_1” to the first instance of a Bootloadable component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance names used in the following tables are “Bootloader” and “Bootloadable.”

Bootloader and Bootloadable Functions

Function	Description
Bootloader_Start()	Once called, a software reset is executed, and then the Bootloader application takes over the CPU.
Bootloader_GetMetadata()	Returns value of the specified field of the metadata section.
Bootloader_ValidateBootloadable()	Verifies validation of the specified application.
Bootloader_Exit()	Schedules the specified application and performs software reset to launch it.
Bootloader_Calc8BitSum()	Computes the 8 bit sum for the specified data.
Bootloadable_Load()	Updates the meta data area for Bootloader to be started on device reset and resets device.

void Bootloader_Start(void)

Description: This function is called in order to execute the following:

- Identify the active bootloadable application (applicable only to the Multi-application bootloader).
- Validate the bootloader application (design-time configurable, Bootloader application validation option of the component customizer).
- Validate the active bootloadable application. If active bootloadable application is not valid, and the other bootloadable application (inactive) is valid, the last one is started.
- Run a communication subroutine (design-time configurable, Wait for command option of the component customizer).
- Schedule the bootloadable and reset the device.

Parameters: None

Return Value: This method will never return. It will either load a new application and reset the device or jump directly to the existing application. The CPU is halted if validation failed when "Bootloader application validation" option is enabled.

PSoC 3/PSoC 5: The CPU is halted if Flash initialization fails.

Side Effects: If the Bootloader application validation option is enabled and this method determines that the bootloader application itself is corrupt, this method will not return. Instead, it will simply hang the application.

uint32 Bootloader_GetMetadata(uint8 field, uint8 appId)

Description: Returns value of the specified field of the metadata section.

Parameters: field: The field to get data from:

Parameter Value	Description
Bootloader_GET_BTLDB_CHECKSUM	Bootloadable Application Checksum
Bootloader_GET_BTLDB_ADDR	Bootloadable Application Start Routine Address
Bootloader_GET_BTLDR_LAST_ROW	Bootloader Last Flash Row
Bootloader_GET_BTLDB_LENGTH	Bootloadable Application Length
Bootloader_GET_BTLDB_ACTIVE	Active Bootloadable Application
Bootloader_GET_BTLDB_STATUS	Bootloadable Application Verification Status
Bootloader_GET_BTLDR_APP_VERSION	Bootloader Application Version
Bootloader_GET_BTLDB_APP_VERSION	Bootloadable Application Version
Bootloader_GET_BTLDB_APP_ID	Bootloadable Application ID
Bootloader_GET_BTLDB_APP_CUST_ID	Bootloadable Application Custom ID

appId: The number of the bootloadable application. Should be 0 for the normal bootloader and 0 or 1 for the Multi-Application bootloader.

Return Value: None

Side Effects: None

cystatus Bootloader_ValidateBootloadable(uint8 appId)

Description: Performs bootloadable application validation by calculating the application image checksum and comparing it with the checksum value stored in the Bootloadable Application Checksum field of the metadata section.

If Fast bootloadable application validation option is enabled in the component customizer and bootloadable application successfully passes validation, the Bootloadable Application Verification Status field of the metadata section is updated.

If Fast bootloadable application validation option is enabled and Bootloadable

Application Verification Status field of the metadata section claims that bootloadable application is valid, the function returns CYRET_SUCCESS without further checksum calculation.

Refer to the Metadata section for the details.

Parameters: appId: The number of the bootloadable application. Should be 0 for the normal bootloader and 0 or 1 for the Multi-Application bootloader.

Return Value: Returns CYRET_SUCCESS if the specified bootloadable application is valid.

Side Effects: None

void Bootloader_Exit(uint8 appId)

Description: Schedules the specified application and performs software reset to launch it.

If the specified application is not valid Bootloader (the result of the ValidateBootloadable() function execution returns other than CYRET_SUCCESS, the bootloader application is launched.

Parameters: application: application to be started.

Constant	Description
Bootloader_EXIT_TO_BTLDR	Bootloader application will be started on software reset.
Bootloader_EXIT_TO_BTLDB, Bootloader_EXIT_TO_BTLDB_1	Bootloadable application # 1 will be started on software reset.
Bootloader_EXIT_TO_BTLDB_2	Bootloadable application # 2 will be started on software reset. Available only if Multi-Application option is enabled in the component customizer.

Return Value: This function never returns.

Side Effects: None

uint8 Bootloader_Calc8BitSum(uint32 baseAddr, uint32 start, uint32 size)

Description: This computes the 8 bit sum for the provided number of bytes contained in Flash (if baseAddr equals CY_FLASH_BASE) or EEPROM (if baseAddr equals CY_EEPROM_BASE).

Parameters: baseAddr:

CY_FLASH_BASE

CY_EEPROM_BASE - applicable only for PSoC 3 / PSoC 5LP devices.

start: The starting address

size: The number of bytes to read and compute checksum

Return Value: Returns 8-bit sum for the provided data

void Bootloadable_Load(void)

Description: Updates the meta data area for Bootloader to be started on device reset and resets device.

Parameters: None

Return Value: None. The processor is reset upon function execution.

Side Effects: None

Sample Firmware Source Code

PSoC Creator provides many example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

See the “Find Example Project” topic in the PSoC Creator Help for more information.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator components
- specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

Bootloader Component Specific Deviations:

Rule	Rule Class	Rule Description	Description of Deviation(s)
14.3	R	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment provided that the first character following the null statement is a white-space character.	Null statement is located close to other code: the CyGlobalIntEnable macro is followed by a semi-colon, while its implementation includes semi-colon. Applicable for PSoC 3/PSoC 5 devices.
14.5	R	The continue statement shall not be used.	A 'continue' statement has been used in 2 places to simplify packet processing.
14.7	R	A function shall have a single point of exit at the end of the function.	Multiple points of exit are used in the function that verifies validity of the bootloadable applications.
19.7	A	A function should be used in preference to a function-like macro.	Deviated since function-like macros are used to allow more efficient code.

Bootloadable Component Specific Deviations:

Rule	Rule Class	Rule Description	Description of Deviation(s)
19.7	A	A function should be used in preference to a function-like macro.	Deviated since function-like macros are used to allow more efficient code.

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.

The measurements were done with the associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

Note For PSoC 4 and PSoC 5LP devices, the SRAM usage is shown without space reserved for heap and stack.

PSoC 3 (Keil_PK51)

Configuration	Flash Bytes	SRAM Bytes
Bootloader	2321	4
Full Bootloader Application [1]	7209	995
Full Bootloadable Application [2]	1561	95

PSoC 4 (GCC)

Configuration	PSoC 4000		PSoC 4100/ PSoC 4200		PSoC 4100-BL/ PSoC 4200-BL	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Bootloader	802	8	802	8	802	8
Full Bootloader Application [1]	4080	356	3808	440	3984	456
Full Bootloadable Application [2]	992	148	886	256	1038	256

1 The measurements for this configuration were done for the entire bootloader project, with the fixed-function based I²C used as communication component and Bootloader component configured for the minimal flash consumption.

2 The measurements for this configuration were done for entire bootloadable project. Note The bootloadable projects contain embedded bootloader projects for the PSoC 4 and PSoC 5LP devices. The size of the bootloadable application is reported.

PSoC 5LP (GCC)

Configuration	Flash Bytes	SRAM Bytes
Bootloader	928	8
Full Bootloader Application [1]	4344	613
Full Bootloadable Application [2]	992	301

Functional Description

Definitions

- **Application Code and/or Data (ACD) –** This file format is installed in flash by the Bootloader.
- **I/O to Flash (IOF) –** Operation performed by Bootloader component, transfer ACD from I/O pins through a communications component to flash.
- **Bootloader Project –** A PSoC Creator project type that can incorporate a Bootloader component.
- **Bootloadable Project –** A PSoC Creator project type that can incorporate a Bootloadable component. It is loaded into flash by a Bootloader in an IOF operation.
- **Bootloader Component –** A component that must be placed onto the schematic of the Bootloader project in order to add bootloader functionality support.
- **Bootloadable Component –** A component that must be placed onto the schematic of the Bootloadable project in order to add bootloadable functionality support.
- **Row –** The Flash data that is accessed in a single operation. Addresses a portion of main flash and of ECC flash (if ECC is available). Actual number of bytes changes based on the selected device. PSoC3/5 has 256 main flash bytes + 32 ECC flash bytes = 288 total bytes per row. PSoC 4 has 128 bytes in a flash row.
- **Error Checking and Correction (ECC) –** The method used to detect and correct errors introduced during data storage. PSoC 3/5 devices support the usage of ECC memory as additional memory when not configured for error protection.
- **Communications component –** Any component that is visible under the Communications node in the Cypress Component Catalog in PSoC Creator, that also implements a standard set of bootloader interface functions.

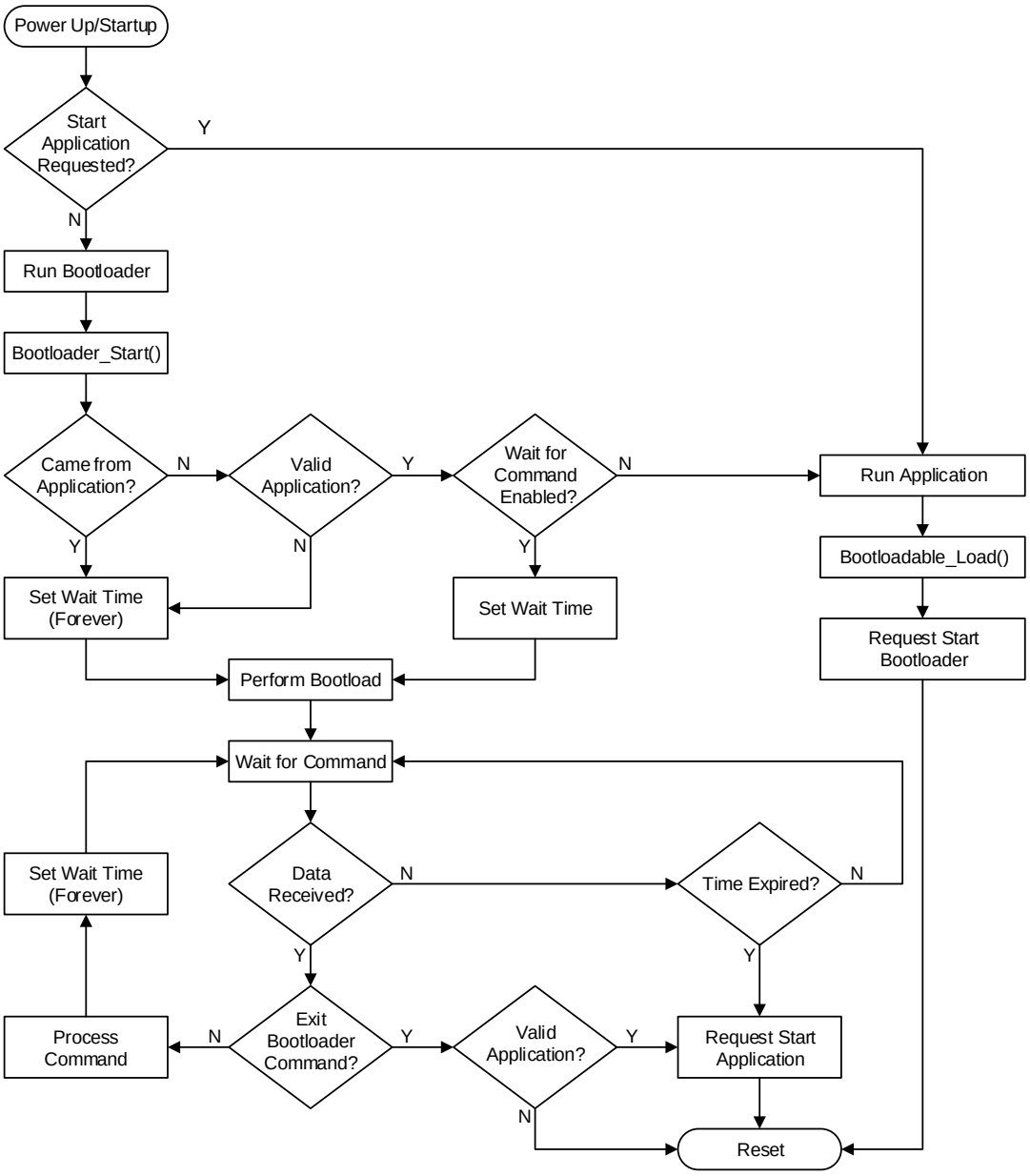
Bootloader and Bootloadable Project Functions

The bootloader project performs overall transfer of a bootloadable project, or new code, to the flash via the bootloader project's communications component. After the transfer, the processor is always reset. The bootloader project is also responsible at reset time for testing certain conditions and possibly auto-initiating a transfer if the bootloadable project is non-existent or is corrupt.

At startup, the bootloader code loads configuration bytes for its own configuration. It must also initialize the stack and other resources as well as peripherals to do the transfer. When the transfer is complete, control is passed to the bootloadable project with a software reset.

The bootloadable project then loads configuration bytes for its own configuration; and reinitializes the stack and other resources and peripherals for its functions. The bootloadable project may call the Bootloadable_Load() function in the bootloadable project to switch to the bootloader application (this results in another software reset).

The following diagram shows how the bootloader works.



When you have finished your development/test cycles and wish to create final images for your bootloader and associated bootloadable applications, make sure to recompile all of the relevant projects using the Release configuration of your IDE.

Bootloader Application

You typically complete a bootloader design project by dragging a Bootloader component and communication component onto the schematic, routing the I/O to pins, setting up clocks, and so on. A project with Bootloader and communication components implements the basic bootloader

application function of receiving new code and writing it to flash. You add custom functions to a basic bootloader project by dragging other components onto the schematic or by adding source code.

Bootloadable Application

The bootloadable application is actually the code. It is very similar to a normal application type. The main differences are that a bootloadable application is always associated with a bootloader application, while a normal project is never associated with a bootloader application.

Export a Design to a 3rd Party IDE

See the "Exporting a Design to a 3rd Party IDE" topic in the PSoC Creator Help for the details on exporting bootloader and bootloadable application to a 3rd party IDE.

Memory Usage

Bootloader

Normal and bootloader applications reside in flash starting at address zero.

Bootloadable

A bootloadable application occupies flash starting from the next empty flash row to the bootloader application.

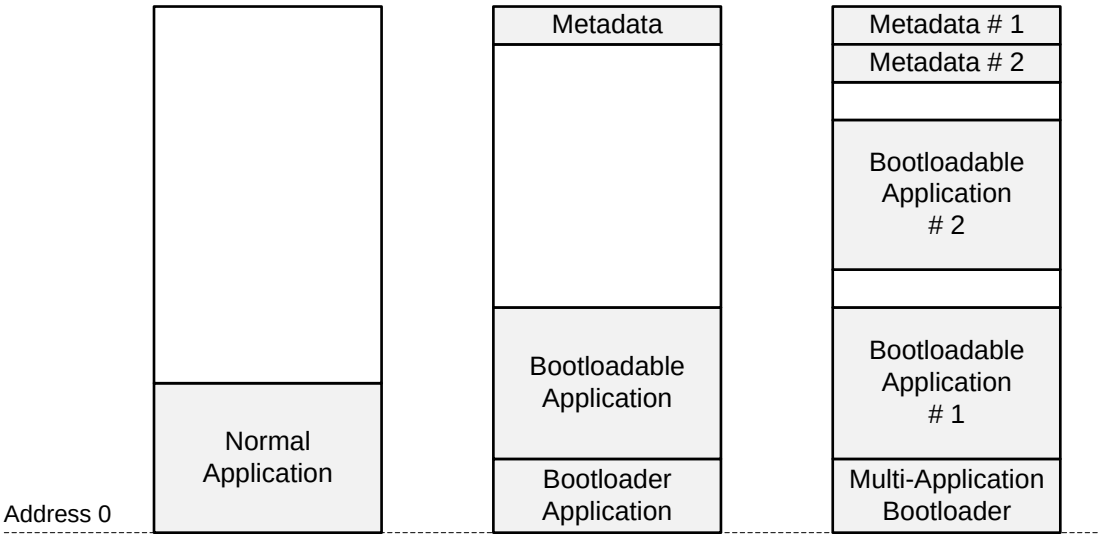
In case of a multi-application bootloader, the first bootloadable application resides above the bootloader application. The second bootloadable application occupies flash starting at the row that is halfway between the start of the first bootloadable application and the end of flash.

If the **Manual application image placement** option in the Bootloadable component customizer is enabled, the bootloadable application is placed at an address specified by the **Placement address** option.

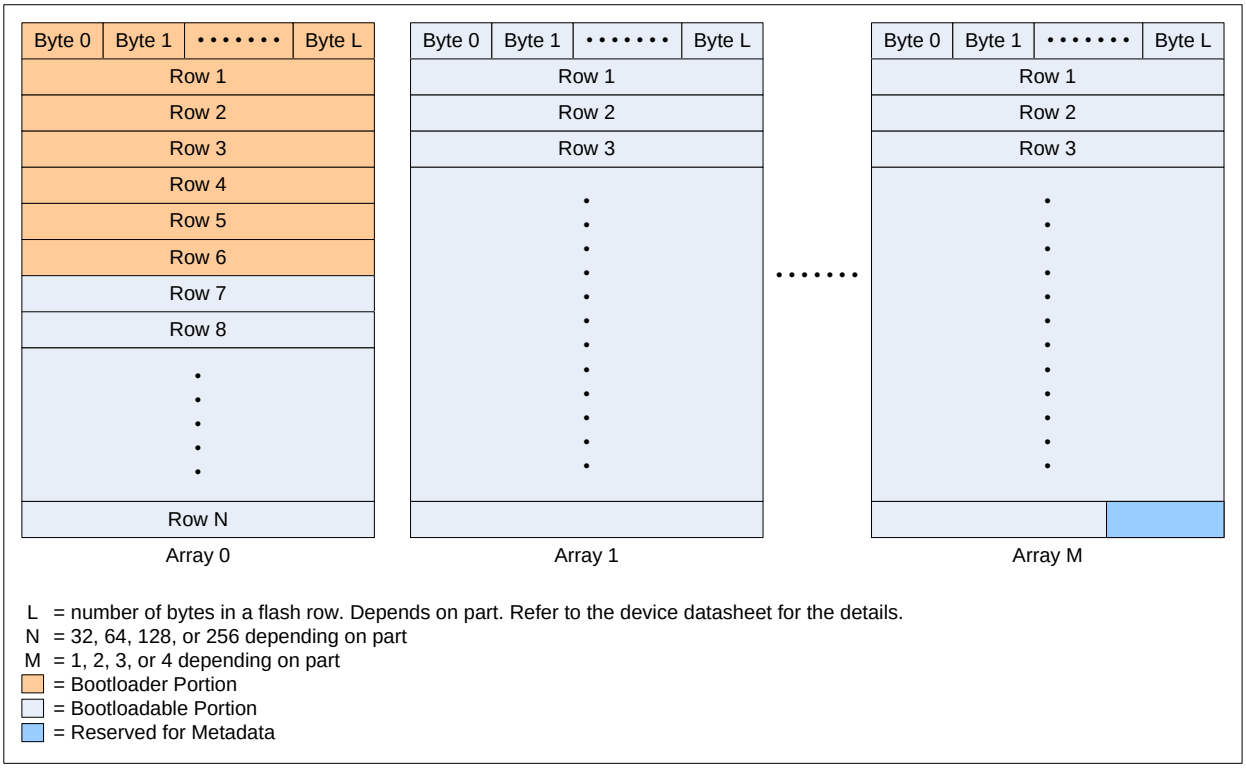
Note In case of a multi-application bootloader, the **Manual application image placement** and **Placement Address** options must be identical for the both bootloadable applications.

Note In case of a multi-application bootloader, the **Manual application image placement** and **Placement Address** options are applicable only to the first bootloadable application. The second bootloadable application occupies flash starting at the row that is halfway between the start of the first bootloadable application and the end of flash.

The following diagram shows (from left to right) the memory usage of normal application, bootloader and bootloadable applications, and the multi-application bootloader two bootloadable applications:



The following diagram shows the device's flash memory layout.



The bootloader project always occupies the bottom X flash rows. X is set so that there is enough flash for:

- The vector table for this project, starting at address 0 (except PSoC 3), and
- The bootloader project configuration bytes, and
- The bootloader project code and data, and
- The checksum for the bootloader portion of flash

The bootloader project configuration bytes are always stored in main flash, never in ECC flash. The relevant option is removed from the bootloader project design-wide resource file.

The bootloader application portion of flash should be protected in the Flash Protection tab of the design-wide resource file to make it only be overwritten by downloading via JTAG / SWD.

The bootloadable project occupies flash starting at the first flash row size boundary after the bootloader, and includes:

- The vector table for the project (except PSoC 3),
- The bootloadable project code and data, and
- 64 bytes of data reserved at the very end of the last flash array to store metadata used by both the bootloader and bootloadable.

The bootloadable project's configuration bytes may be stored in the same manner as in a standard project, that is, in either main flash or in ECC flash, per settings in the design-wide resource file.

Device-specific Details

PSoC 3

In the PSoC 3, the only "exception vector" is the 3-byte instruction at address 0, which is executed at processor reset. (The interrupt vectors are not in flash – they are supplied by the Interrupt Controller [IC]). So at reset the PSoC 3 bootloader code simply starts executing from flash address 0.

PSoC 5LP and PSoC 4

In the PSoC 5LP / PSoC 4 devices, a table of exception vectors must exist at address 0. (The table is pointed to by the Vector Table Offset Register, at address 0xE000ED08, whose value is set to 0 at reset.) The bootloader code starts immediately after this table.

The table contains the initial stack pointer (SP) value for the bootloader project, and the address of the start of the bootloader project code. It also contains vectors for the exceptions and interrupts to be used by the bootloader.

The bootloadable project also has its own vector table, which contains that project's starting SP value and first instruction address. When the transfer is complete, as part of passing control to the bootloadable project the value in the Vector Table Offset Register is changed to the address of the bootloadable project's table.

Metadata Memory Map

The metadata section is a 64-byte block of flash that is used as a common area for both bootloader and bootloadable applications. In the bootloader application, the metadata is placed at row N-1; in case of multiapplication bootloader, the bootloadable application number 1 uses row N-1, and application number 2 uses row N-2 to store its metadata, where N is the total number of rows for the selected device.

Address	PSoC 3	PSoC 4 / PSoC 5LP
0x00	Bootloadable Application Checksum	
0x01	Reserved	Bootloadable Application Start Routine Address
0x02		
0x03	Bootloadable Application Start Routine Address	
0x04		
0x05	Reserved	Last Bootloader Row
0x06		
0x07	Last Bootloader Row	Reserved
0x08		
0x09	Reserved	Bootloadable Application Length
0x0A		
0x0B	Bootloadable Application Length	
0x0C		
0x0D	Reserved	
0x0E		
0x0F		
0x10	Active Bootloadable Application	
0x11	Bootloadable Application Verification Status	
0x12	Bootloader Application Version	
0x13		
0x14	Bootloadable Application ID	
0x15		

Address	PSoC 3	PSoC 4 / PSoC 5LP
0x16	Bootloadable Application Version	
0x17		
0x18	Bootloadable Application Custom ID	
0x19		
0x1A		
0x1B		
0x1C- 0x3F	Reserved	

Name	Description
Bootloadable Application Checksum	This is the basic summation checksum that is computed by adding all the bytes of the bootloadable application image (excluding the metadata section).
Bootloadable Application Start Routine Address	Startup routine address of the bootloadable application. This is STARTUP1 for PSoC 3 and Reset() for PSoC 4 / PSoC 5. The linker is free to put these anywhere it wants after the minimum starting address of the application.
Bootloader Last Flash Row	The number of the last flash row occupied by bootloader application image. **Note** For the second bootloadable application (in the Multi-Application Bootloader case), this field contains last flash row occupied by the first bootloadable application.
Bootloadable Application Length	The size of the bootloadable application in bytes.
Active Bootloadable Application	This field contains information about active bootloadable application if **Multi-application bootloader** option is enabled.
Bootloadable Application Verification Status	This field contains the status of the bootloadable application validation when **Fast bootloadable application validation** option is enabled: the bootloader only computes the checksum the first time and assumes that it remains valid in each future startup.
Bootloader Application Version	This field contains the application version of the bootloader application. Specified in the bootloader component customizer.
Bootloadable Application ID [3]	This field contains the application ID of the bootloadable application. Specified in the bootloadable component customizer.
Bootloadable Application Version [4]	This field contains the application version of the bootloadable application. Specified in the bootloadable component customizer.

3 When the bootloader application is the only application in the device (no bootloadable applications are stored), this field reports the number of images the bootloader application expects; 1 for bootloader projects, and 2 for multi-application bootloader projects.

4 For PSoC 3/PSoC 5LP, when the bootloader application is the only application in the device, this field reports whether ECC memory data should be included in the bootloadable application checksum.

Name	Description
Bootloadable Application Custom ID	This field contains the application custom ID of the bootloadable application. Specified in the bootloadable component customizer.

Note All fields are stored in the endianness of the processor: big-endian for PSoC 3 and little-endian for PSoC 4/PSoC 5LP.

PSoC Creator Project Output Files

When either project type – bootloader or bootloadable - is built, an output file is created for that project.

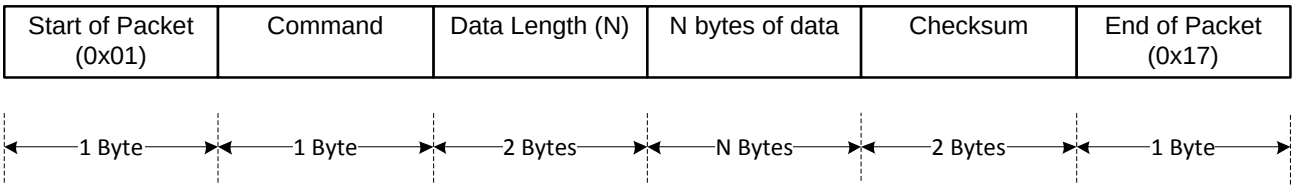
In addition, an output file for both projects – a "combination" file – is created when the bootloadable project is build. This file includes both the bootloader and bootloadable projects. This file is typically used to facilitate downloading both projects (via JTAG / SWD) to device flash in a production environment.

Configuration bytes for bootloadable projects may be stored in either main flash or in ECC flash. The format of the bootloadable project output file is such that when the device has ECC bytes which are disabled, transfer operations are executed in less time. This is done by interleaving records in the bootloadable main flash address space with records in the ECC flash address space. The bootloader takes advantage of this interleaved structure by programming the associated flash row once – the row contains bytes for both main flash and ECC flash.

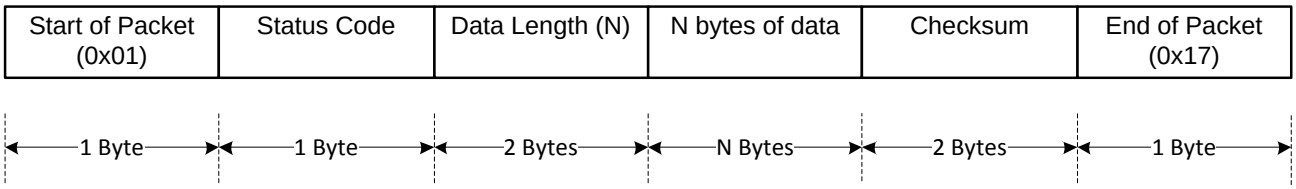
Each project has its own checksum. The checksums is included in the output files at project build time.

Bootloader Packet Structure

Communication packets sent from the Host to the Bootloader have this structure:



Response packets read from the Bootloader have this structure:



Status/Error Codes

The possible status/error codes output from the bootloader are:

Status/Error Code	Value	Description
CYRET_SUCCESS	0x00	The command was successfully received and executed
BOOTLOADER_ERR_LENGTH	0x03	The amount of data available is outside the expected range
BOOTLOADER_ERR_DATA	0x04	The data is not of the proper form
BOOTLOADER_ERR_CMD	0x05	The command is not recognized
BOOTLOADER_ERR_CHECKSUM	0x08	Packet checksum does not match the expected value
BOOTLOADER_ERR_ARRAY	0x09	Flash array ID is not valid
BOOTLOADER_ERR_ROW	0x0A	The flash row number is not valid
BOOTLOADER_ERR_APP	0x0C	The application is not valid and cannot be set as active
BOOTLOADER_ERR_ACTIVE	0x0D	The application is currently marked as active
BOOTLOADER_ERR_UNK	0x0F	An unknown error occurred

Bootloader Commands

The bootloader supports these commands. All received bytes that do not start with one of the set of command bytes is discarded with no response generated. All multi-byte fields are output LSB first.

Note The time required for the bootloader to execute any command is based on the configuration of the device. Some of the factors that affect the timing include:

- Clock speed at which the part is running
- Toolchain used to build the project
- Optimization settings used during the build
- Number of interrupts running in the background

Bootloader Command Name (Command Code)			
Data Byte (bytes number)	**Response Packet Status Code**	**Response Packet Data (bytes number)**	**Description**
Enter Bootloader (0x38)			
N/A	Success Error Command Error Data Error Length Error Checksum	Silicon ID (4) Silicon Rev (1) Version (3)	The bootloader responds to this command with the device information and version of the Bootloader component. Version means version of the Bootloader component.
Get Flash Size (0x32) (optional)			
Flash Array ID (1)	Success Error Command Error Data Error Length Error Checksum	First available row (2) Last available row (2)	The bootloader responds to this command with the first full row after the bootloader application (first row of the bootloadable application) and last flash row in the selected flash array: For the flash array where the bootloader application ends, the first full row after the bootloader application is returned. For the fully occupied flash array, the number of rows in the array plus one is returned (there is no space for the bootloadable application in this array). For the arrays next to the occupied array, zero is returned (bootloadable application can be written from their beginning).
Program Row (0x39)			
Flash Array ID (1) Flash Row Number (2) Data to write (n)	Success Error Command Error Data Error Length Error Checksum Error Flash Row Error Active	N/A	Writes one row of flash data to the device. The data to be written to the flash can be sent in multiple packets using the Send Data command. This command may be sent along with the last block of data, to program the row.
Erase Row (0x34) (optional)			

Bootloader Command Name (Command Code)			
Data Byte (bytes number)	**Response Packet Status Code**	**Response Packet Data (bytes number)**	**Description**
Flash Array ID (1) Flash Row Number (2)	Success Error Command Error Data Error Length Error Checksum Error Flash Row Error Active	N/A	Erases the contents of the provided flash row.
Verify Row (0x3A) (optional)			
Flash Array ID (1) Flash Row Number (2)	Success Error Command Error Data Error Length Error Checksum	Row checksum (1)	Gets a 1 byte checksum for the contents of the provided row of flash.
Verify Checksum (0x31)			
N/A	Success Error Command Error Data Error Length Error Checksum	Checksum valid (1)	A non-zero return value indicates that the application code flash checksum matches the expected value stored in flash and therefore the application is valid. A return value of 0 indicates that the checksums do not match, and therefore the application is not valid.
Send Data (0x37) (optional)			
Data for Device (n)	Success Error Command Error Data Error Length Error Checksum	N/A	Sends a block of data to the device. This data is buffered up in anticipation of another command that will inform the bootloader what to do with the data. If multiple send data commands are issued back-to-back, the data is appended to the previous block. This command is used to breakup large transfers into smaller pieces to prevent bus starvation in some protocols.
Sync bootloader (0x35) (optional)			
N/A	N/A	N/A	Resets the bootloader to a clean state, ready to accept a new command. Any data that was buffered is thrown out. This command is only needed if the host and client get out of sync with each other.

Bootloader Command Name (Command Code)			
Data Byte (bytes number)	**Response Packet Status Code**	**Response Packet Data (bytes number)**	**Description**
Exit Bootloader (0x3B)			
N/A	N/A	N/A	Exits from the bootloader by triggering software reset of the device. Before the software reset is executed, the bootloadable application is verified. If the application passes verification, the application will be executed after the software reset. If the application fails verification, then execution will begin again with the bootloader after the software reset.
Get Metadata (0x03C) (optional)			
Application # (1)	Success Error Application Error Length Error Data Error Checksum	Metadata (56)	Reports first 56 bytes of the metadata for a selected application. For more information on metadata see Metadata section.
Get Application Status (Multi-application bootloader Only) (0x33) (optional)			
Application # (1)	Success Error Length Error Checksum Error Data	App # Valid (1) App # Active (1)	Returns the status of the specified application.
Set Active Application (Multi-application bootloader Only) (0x36)			
Application # (1)	Success Error Application Error Length Error Data Error Checksum	N/A	The specified bootloadable application is set as active. This command is used to switch between two bootloadable applications.

Bootloader Application and Code Data File Format

The bootloader application and code data (.cyacd) file format stores the bootloadable portion of a design. The file is a header followed by lines of flash data. Excluding the header, each line in the .cyacd file represents an entire row of flash data. The data is stored as ASCII data in big endian format.

The header record has this format:

[4-byte SiliconID][1-byte SiliconRev][1-byte Checksum Type]

The data records have this format:

[1-byte ArrayID][2-byte RowNumber][2-byte DataLength][N-byte Data][1-byte Checksum]

The checksum type in the header indicates the type of checksum used for packets sent between the bootloader host and the bootloader itself. The checksum in the data records is a basic summation, computed by summing all bytes (excluding the checksum itself) and then taking the 2's complement.

Bootloader Host Tool

PSoC Creator ships with a bootloader host tool (*bootloader_host.exe*) that you can use to test the bootloader running on a PSoC chip. The bootloader host tool is the application that communicates directly with the bootloader to send new bootloadable images. The bootloader host tool provided is only a development and testing tool.

Source Code

In addition to the host executable itself, much of the source code used is also provided. Use this source code to create your own bootloader host applications. The source code is located in this directory:

*<Install Dir>**cybootloaderutils*\

By default, this directory is:

C:\Program Files\Cypress\PSoC Creator\<Release Version>\PSoC Creator\cybootloaderutils\

This source code is broken up into four different modules. These modules provide implementations for the various pieces of functionality required for a bootloader host. Depending on the desired level of control, some or all of these modules can be used in developing a custom bootloader host application.

cybtldr_command.c/h

This module handles construction of packets to send to the bootloader, and the parsing of packets received from the bootloader. It has a single function for constructing each type of packet that the bootloader understands, and a single function for parsing the results for each packet the bootloader can send back.

cybtldr_parse.c/h

This module handles the parsing of the *.cyacd file that contains the bootloadable image to send to the device. It has functions for Setting up access to the file, Reading the header, Reading the row data, and closing the file.

cybtldr_api.c/h

This is a row level API that allows for sending a single row of data at a time to the bootloader using a supplied communication mechanism. It has functions for setting up the bootload

operation, programming a row, erasing a row, verifying a row, and ending the bootload operation.

cybtldr_api2.c/h

This is a higher level API that handles the entire bootload process. It has functions for programming the device, erasing the device, verifying the device, and aborting the current operation.

Resources

The Bootloader and Bootloadable projects use these device resources:

- The Bootloader component uses both general purpose bits of the reset status (RESET_SR0) register. These bits are necessary to communicate bootloader intents across the software reset boundaries.
- The resources used by communication component are in the corresponding component datasheet.

Component Errata

This section lists known problems with the Bootloader component.

Cypress ID	Component Version	Problem	Workaround
224638	1.30, 1.40	The "Fast bootloable application validation" option does not work for PSoC 3 and PSoC 5LP devices.	This defect is fixed in version 1.50. If updating the Bootloader component to version 1.50 is not an option, add a CySetTemp() API call before starting the Bootloader component.

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.30.a	Updated datasheet.	Added errata section to document defect 224638.
1.30	Updated list of the supported communication components.	To provide updated interface information.
	Aligned diagram in the Bootloader and Bootloadable Project Functions section with the implementation.	Bootloadable application validation is performed from the bootloader application before switching to it through the software reset.

Version	Description of Changes	Reason for Changes / Impact
	Added support for Bluetooth Low Energy devices.	
	Added following functions to the Bootloader component: uint32 Bootloader_GetMetadata(uint8 field, uint8 appId) cystatus Bootloader_ValidateBootloadable(uint8 appId) void Bootloader_Exit(uint8 appId) uint8 Bootloader Calc8BitSum(uint32 baseAddr, uint32 start, uint32 size)	Increased functionality.
	Updated Bootloader_Start() for the Multi-Application Bootloader.	To implement the following algorithm: If active bootloadable application is not valid, and the other bootloadable application (inactive) is valid, the last one is started.
	Fixed an issue when Verify Row command was always available independently of the customizer settings.	
	Ensured that the bootloader application is not overwritten during bootloadable application transfer.	Implemented additional verification.
	The metadata flash row of the active bootloadable application is erased before the process of updating bootloadable application is launched only if “Fast bootloadable application validation” option is enabled. The feature was added for the Multi-Application Bootloader projects.	The metadata flash row was erased independently of the “Fast bootloadable application validation”. The reason of erasing metadata flash row is to reset “Bootloadable Application Verification Status” field and trigger checksum computation instead of relying on status stored in the metadata.
	Updated the Get Flash Size command implementation.	To address incorrect reply when bootloader application consumes more than one flash array (its size is above 64 KB).
	The flash initialization for the PSoC 3 and PSoC 5LP devices updated to be performed only before flash write.	The startup time (time between reset and main() entry) significantly decreased.
1.20.a	Minor datasheet edits.	Added note for PSoC 4000 devices and flash.
1.20	The Wait for command time option was changed to be in units of 100 ms instead of 10 ms units.	**Note** While updating to version 1.20 the Wait for command time option value will be automatically increased by 10 times.
	Added Get Metadata command.	Reports first 56 bytes of the metadata for a selected application.

Version	Description of Changes	Reason for Changes / Impact
	All commands (with the exception of Exit Bootloader, and Sync Bootloader) are ignored by Bootloader application till the Enter Bootloader command is received.	Bootloader application waits for valid traffic (denoted by Enter Bootloader command), but not for any traffic. If traffic is received but not a valid bootloader Enter Bootloader command, then the timeout expires at the specified time and the bootloadable application is launched.
	Updated the **Dependencies** tab.	Added field to specify Bootloader ELF file.
	Updated MISRA Compliance section.	The Bootloader/Bootloadable components were verified for MISRA compliance and have specific deviations described.
1.10	Added MISRA Compliance section.	The Bootloader/Bootloadable components were not verified for MISRA compliance.
	Added PSoC 4 device support.	New device support
	Minor datasheet edits	
1.0.a	Datasheet corrections	
1.0	Initial component version	

© Cypress Semiconductor Corporation, 2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.