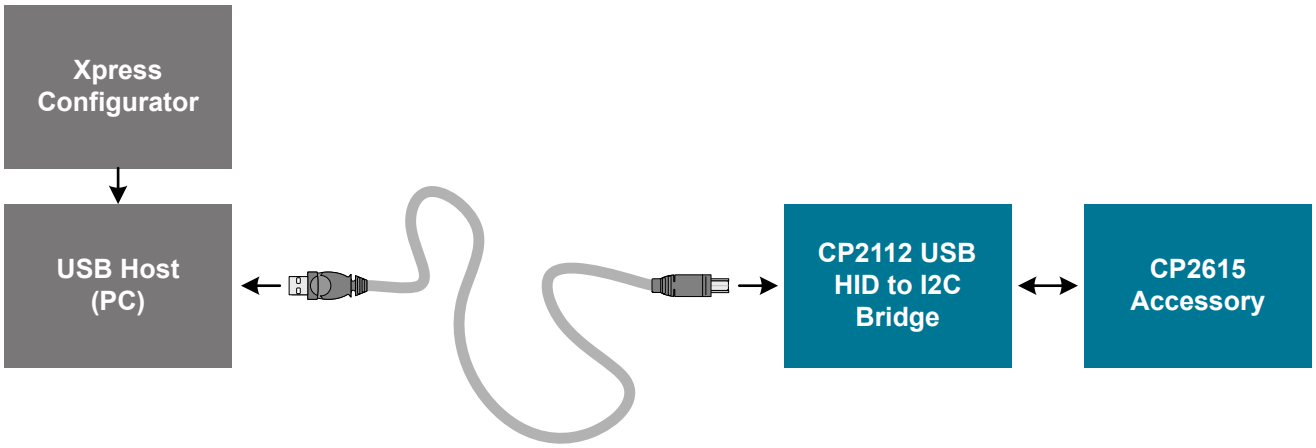
AN1044: CP2615 Customization User Guide

This document explains the options that are available for customization on CP2615 fixed function USB devices.

It contains information about obtaining a Vendor ID (VID) and Product ID (PID) for a CP2615 product and describes the steps necessary for customizing the device descriptors using Xpress Configurator within Simplicity Studio (http://www.silabs.com/simplicity). For a more detailed description of Xpress Configurator operation, please refer to *AN721: CP210x/CP211x Device Customization Guide* http://www.silabs.com/products/Interface/Pages/interface-application-notes.aspx

KEY POINTS

- This document describes how to obtain and customize the VID, PID, and user identification strings for a CP2615-based product.
- Customize the CP2615:
  - Audio Interface
  - GPIOs
  - CODEC configuration



1. USB Vendor IDs and Product IDs

Each type of audio accessory that is connected to a USB host device must have a unique Vendor ID (VID), Product ID (PID), and serial number combination. This ID system uniquely identifies the different devices on the bus to avoid conflicts. The VID/PID must be unique in that each USB device with the same VID/PID will use the same driver, and it is strongly recommended to make the PID unique to a particular design. The USB devices of a given VID/PID combination can be serialized, which allows the operating system to track not only a particular model, but also a specific board of that model.

Vendor IDs are owned by the vendor company and assigned by the USB Implementers Forum (USB-IF) only. Details about obtaining a unique VID can be found at www.usb.org/developers/vendor. To obtain the right to license the USB-IF logo, register the product's VID and PID with USB-IF and submit the product to the USB-IF Compliance Program. USB-IF Compliance Program details are available at www.usb.org/developers/compliance. Once the product is certified, it can be added to the USB-IF Integrators List, and the “Certified USB” logo can be used on the product.

2. Device Customization Software

The CP2615 has a number of properties that can be selected and changed by the customer. Simplicity Studio (http://www.silabs.com/simplicity) provides a tool, Xpress Configurator, to select the property configuration and to program it into a CP2615. It uses a USB connection to a CP2112 as the programming interface for the CP2615, via I^2C. Please see the *CP2615-EK User's Guide* for the board schematic that demonstrates this connection: http://www.silabs.com/support/resources.ct-manuals.p-interface_usb-bridges.

The CP2615 configuration can also be set in the factory at production time for large orders. Contact your Silicon Laboratories sales representative for details.

The CP2615 customization properties are organized into groups in the properties pane. The properties available in each group are described in the following sections.

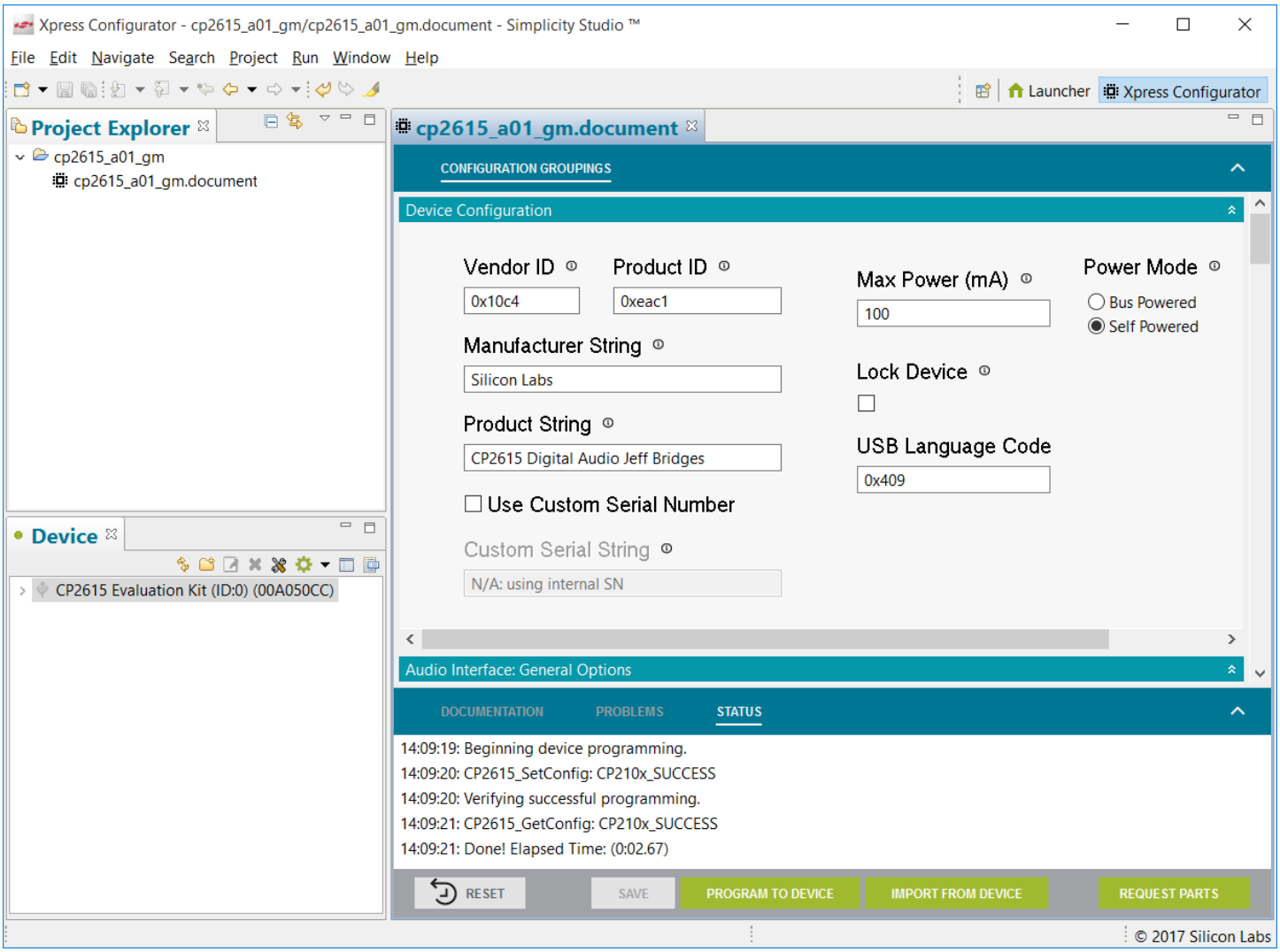


Figure 2.1. CP2615 Xpress Configurator

2.1 Device Options

The first group of properties is related to accessory identification and power-related features of the CP2615-based accessory. The accessory may be powered from a separate supply such as batteries or a wall plug power supply. In this case the accessory can be "self-powered" and has no power interaction with the host. If the accessory draws power from the host, even if it also has its own power supply, then the [**Bus Powered**] option should be chosen for the [**Power Mode**] property.

Table 2.1. Device Configuration Options

Property	Description	Values
Vendor ID (hex)	The Vendor ID is a four hexadecimal digit number such as 10C4. Each CP2615-based accessory type must use a unique VID/PID combination to identify the organization and product. More information on how to obtain a VID and PID can be found in 1. USB Vendor IDs and Product IDs.	16-bit hexadecimal
Product ID (hex)	The Product ID is a four hexadecimal digit number such as EAB0. Each CP2615 application must use a unique VID/PID combination to identify the organization and product. More information on how to obtain a VID and PID can be found in 1. USB Vendor IDs and Product IDs.	16-bit hexadecimal
Use Custom Serial Number	The CP2615 has a factory programmed serial number which will be used as the accessory serial number by default. This can be overridden and a custom serial number provided as part of the configuration. If set to [**No**], then the factory serial number will be used. If [**Yes**], then the serial number defined by Serial (below) will be used.	• No • Yes
Custom Serial String	This is the custom serial string used if [**Use Custom Serial Number**] is set to [**Yes**].	The serial number string is UTF-8 and can be any combination of characters. The total length cannot exceed 30 bytes. Note that UTF-8 characters may be more than 1 byte.
Manufacturer String	This is the name of the company manufacturing the product.	UTF-8 string of any length or character combination
Product Description	Usually this is text which provides a description of the device, such as CP2615 Accessory Audio Bridge EVB.	UTF-8 string of any length or character combination
Power Mode	This selects between self powered or bus powered mode. If the accessory has its own power supply, then choose [**Self Powered**]. If the accessory does not have its own power supply or draws any power from the host, then choose [**Bus Powered**].	• Bus Powered • Self Powered
Maximum Power (mA)	The highest amount of current that the accessory will draw from the host. The maximum allowed is 100 mA. If the accessory is self powered and does not draw any power from the host, then this value should be 0.	integer value between 0 and 100, units of mA
Lock Device	Locks the device from all further configuration. The device will no longer respond to read or write configuration commands.	• False • True
USB Language Code (hex)	This USB language code used for USB strings. The CP2615 only supports one configurable language code at a time, for example: 0x0409 for English.	16-bit hexadecimal

2.2 Audio Interface Options

Table 2.2. Audio Interface Configuration Options

Property	Description	Values
Audio Interfaces	Chooses the streaming audio configuration.	• No Audio • Audio In (16 bit) • Audio In (24/16 bit) • Audio Out (16 bit) • Audio Out (24/16 bit) • Audio In/Out (16 bit) • Audio In (24 bit only) • Audio Out (24 bit only)
Clock Sync Mode	Chooses the clock synchronization mode.	• Synchronous • Asynchronous
Audio Sample Rates	Chooses the audio sample rates in kHz to be supported by the audio accessory.	• 44.1 • 48 • 48/44.1
Feature Unit	The feature unit allows the USB host to control codec volume and mute. When the feature unit is [**Enabled**], the USB host can send volume and mute commands that the CP2615 will turn into commands to the codec to adjust the volume or mute/unmute. This is the correct setting for most codecs. However, some codecs do not have the ability to control volume or mute. In this case the feature unit should be [**Disabled**] and the USB host performs volume control by scaling the audio samples.	• Disabled • Enabled
Input Terminal Type	Chooses the input terminal type that is used to identify the audio device to the host. The choice of terminal type does not affect the behavior of the CP2615, but may make a difference to the audio capabilities available to the host.	• Microphone • Line In
Output Terminal Type	Chooses the output terminal type that is used to identify the audio device to the host. The choice of terminal type does not affect the behavior of the CP2615, but may make a difference to the audio capabilities available to the host.	• Headphones • Speakers • Line Out

Property	Description	Values
I2S_MCLK Active	Controls whether the I²S master clock runs continuously or turns off when there is no audio streaming. Allowing the MCLK to turn off when not streaming will normally allow for power saving. But there may be some hardware or codec configuration that requires MCLK to remain running.	• Only When Streaming • Always
I2S_LRCLK Active	This setting is similar to I2S_MCLK Active, above.	• Only When Streaming • Always
The correct values for the following properties are dependent on the characteristics of the codec chosen for use with the CP2615.		
Volume Master Default (dB)	Initial playback master volume setting.	8-bit integer dB
Volume Left Default (dB)	Initial playback left channel volume setting.	8-bit integer dB
Volume Right Default (dB)	Initial playback right channel volume setting.	8-bit integer dB
Volume Min (dB)	Minimum allowable playback volume setting reported for the feature unit.	8-bit integer dB
Volume Max (dB)	Maximum allowable playback volume setting reported for the feature unit.	8-bit integer dB
Volume Resolution (dB)	Volume adjustment resolution reported of the feature unit.	decimal, units of dB
Volume Min Counts	Codec volume register value that corresponds to the minimum volume setting in dB.	8-bit integer
Volume Max Counts	Codec volume register value that corresponds to the maximum volume setting in dB.	8-bit integer

2.3 Codec Options

The properties in this section are related to control of the codec. The values chosen here depend on the register interface and characteristics of the codec. The I2C strings are a set of instructions that are executed at certain times and are intended to program the codec for different modes. For example, there is a command string used to initialize the codec, and another that is used for setting volume. The contents of the I2C command strings will be different for each kind of codec. There is no fixed limit on the length of any one I2C string; however, the total length of the entire configuration is limited. When programming the accessory, the customization utility will display the configuration size and provide a warning if the configuration is over the total limit. For more information about the I2C strings, please see the CP2615 data sheet.

Table 2.3. Codec Configuration Options

Property	Description	Values
Register Size	Size of codec registers, in bytes.	• 1 byte • 2 bytes
Register Format	Codec volume register format.	• Unsigned • Signed
Volume Bit Start Position	Offset of volume bit field within the volume register.	0-15
Volume AND Mask	Mask applied before writing the volume register. This can be used to force other bits in the volume register to 0.	16-bit hex
Volume OR Mask	Mask applied before writing the volume register. This can be used to force other bits in the volume register to 1.	16-bit hex
Playback Mute via Register	If yes, playback muting is performed using I2C "Set Mute" strings (see below).	• No • Yes
Playback Mute Register Polarity	Determines the polarity of the mute bit written to the corresponding register.	• Mute when Low • Mute when High
Playback Mute Register Mask	Bit mask of mute bits within the corresponding register.	2-byte, left justified
I2C Startup Delay (ms)	Delay from entering active mode to initializing the codec. Used to allow codec power up if codec power is switched.	integer ms
I2C String: Codec Initialize	I2C command string to be executed when entering High Power mode.	array of bytes
I2C String: Codec High-> Low	I2C command string to be executed when entering Low Power mode.	array of bytes
I2C String: Audio Start	I2C command string to be executed when audio streaming (playback or record) first begins.	array of bytes
I2C String: Audio Stop	I2C command string to be executed when all audio streaming (playback and record) has stopped.	array of bytes
I2C String: L-Volume Prefix	Portion of I2C command string that precedes the byte(s) that comprise the Left channel volume setting. This byte array should not be zero-terminated.	array of bytes
I2C String: L-Volume Suffix	Portion of I2C command string that follows the byte(s) that comprise the Left channel volume setting.	array of bytes
I2C String: R-Volume Prefix	Portion of I2C command string that precedes the byte(s) that comprise the Right channel volume setting. This byte array should not be zero-terminated.	array of bytes
I2C String: R-Volume Suffix	Portion of I2C command string that follows the byte(s) that comprise the Right channel volume setting.	array of bytes
I2C String: Get Mute Prefix	I2C bytes to be transmitted prior to reading the codec register byte(s) that contain the current mute setting.	array of bytes

Property	Description	Values
I2C String: Set Mute Prefix	I2C bytes to be transmitted prior to writing the mute setting byte(s) to the codec. This byte array should not be zero-terminated.	array of bytes
I2C String: Set Mute Suffix	I2C bytes to be transmitted after writing the mute setting byte(s) to the co-dec.	array of bytes
I2C String: Set Rate 44.1	I2C command string to be executed when host sets sample rate to 44.1 kHz.	array of bytes
I2C String: Set Rate 48.0	I2C command string to be executed when host sets sample rate to 48 kHz.	array of bytes

2.3.1 I2C Command Interpreter

Xpress Configurator in Simplicity Studio (http://www.silabs.com/simplicity) includes a command interpreter to generate the appropriate I2C command byte arrays needed by the device to operate the codec.

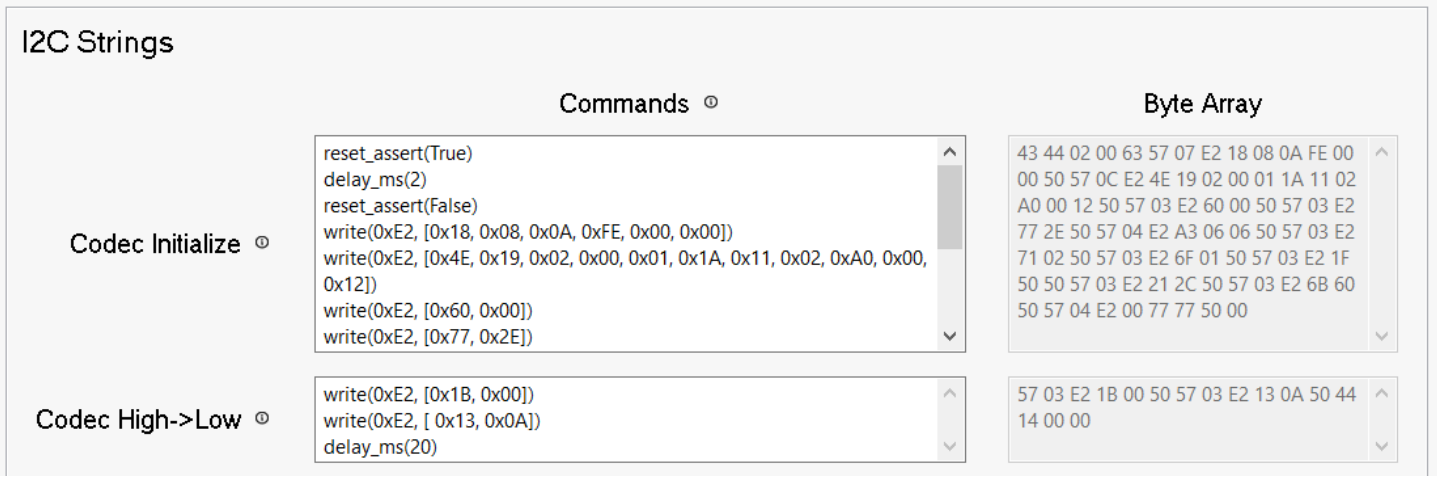


Figure 2.2. I2C Command Interpreter

2.3.1.1 Commands

The commands that are available to the interpreter are:

2.3.1.1.1 write

Description : This function writes an array of bytes over I2C to the designated slave address.

Prototype : `write(slaveAddr, data, i2c_stop=True)`

Parameters :

1. slaveAddr—Slave address of the codec [byte]
2. data—A bracketed array of data to send to the slave. [array of comma-delimited bytes]

   Note: Volume I2C commands must include one, and only one, write command with *VOLUME* in the data parameter.

   Note: The Set Mute I2C command must include one, and only one, write command with *MUTE* in the data parameter.

3. i2c_stop (optional)—Determines whether or not to end the I2C transaction with an I2C stop condition. Set to *False* and follow with another I2C transaction for a repeated-start. If not specified, this defaults to *True*. [*True*,*False*]

Examples : `write(0xE2, [1, 2, 3])`— Write the values "1", "2", and "3" to the I2C slave with address "0xE2", ending the I2C transaction with an I2C stop condition.

`write(0xE2, [0x12, 0x34], i2c_stop=False)`— Write the values "0x12" and "0x34" to the I2C slave with address "0xE2", without ending the I2C transaction with an I2C stop condition.

2.3.1.1.2 read

Description : This function reads a specified number of bytes over I2C to the designated slave address.

Prototype : `read(slaveAddr, numBytes, i2c_stop=True)`

Parameters :

1. slaveAddr—Slave address of the codec. [byte]
2. numBytes—Number of bytes to read from the slave. [byte]
3. i2c_stop (optional)—Determines whether or not to end the I2C transaction with an I2C stop condition. Set to *False* and follow with another I2C transaction for a repeated-start. If not specified, this defaults to *True*. [*True*,*False*]

Examples : `read(0xE2, 5)`— Reads 5 bytes from the slave with address "0xE2", ending the I2C transaction with an I2C stop condition.

`write(0xE2, 2, i2c_stop=False)`— Reads 2 bytes from the I2C slave with address "0xE2", without ending the I2C transaction with an I2C stop condition.

2.3.1.1.3 reset_assert

Description : This function asserts or deasserts the RESETOUTb line that interfaces with the codec.

Prototype : `reset_assert(assertReset)`

Parameters :

1. assertReset—Determines whether or not to assert the RESETOUTb pin. Set to *True* to assert RESETOUTb, set to *False* to deassert RESETOUTb. [*True*,*False*]

Examples : `reset_assert(True)`— Assert RESETOUTb

`reset_assert(False)`— Deassert RESETOUTb

2.3.1.1.4 delay_ms

Description : Delay the execution of commands by a given number of milliseconds

Prototype : `delay_ms(delay_ms)`

Parameters : 1. delay_ms—Number of milliseconds to delay. [byte]

Examples : `delay_ms(1)`— Delay for 1 millisecond

`delay_ms(255)`— Delay for 255 milliseconds

2.3.1.1.5 reboot

Description : Reboot the CP2615

Prototype : `reboot(waitForTransactionComplete)`

Parameters : 1. waitForTransactionComplete—Determines whether to reboot the CP2615 instantly, or if the CP2615 should complete the current transaction before rebooting. [*True*,*False*]

Examples : `reboot(True)`— Reboot after the current transaction is completed

`reboot(False)`— Reboot immediately

2.3.1.2 Operation

The interpreter operates by parsing one command per line and generating the appropriate byte array for the CP2615. For example, the default setting for the [**Codec High->Low**] command string is:

```
write(0xE2, [0x1B, 0x00])
write(0xE2, [ 0x13, 0x0A])
delay_ms(20)
```

This generates the following byte array:

```
57 03 E2 1B 00 50 57 03 E2 13 0A 50 44 14 00 00
```

2.3.1.2.1 Input Values

Byte

A byte is a number that can have a value from 0 to 255 (0x00 to 0xFF). For inputs to commands that are bytes (slave address, delay in milliseconds, data values), the input can be represented in decimal or hexidecimal. For example, to delay 20 milliseconds, you could enter either of the following commands:

```
delay_ms(20)
delay_ms(0x14)
```

Bool

For boolean type parameters, the input value can be *True* or *False*, or a number of values that will evaluate to True or False, such as zero (False), a non-zero number (True), or *None* (False). For example:

```
reset_assert(True)
reboot(False)
```

Data Array

The 2.3.1.1.1 write command has a parameter called *data* that is an array of byte values. This array is represented by a bracketed, comma-delimited list of bytes. These bytes follow the same rules as above, so they can be represented in decimal or hexidecimal (if prefixed by '0x'). For example, this is a valid write command with a data array:

```
write(0xE2, [0x18, 0x08, 0x0A, 254, 0, 0])
```

For two specific cases, the write command can also contain a keyword.

For the [**Set L-Volume**] and [**Set R-Volume**] command strings, the commands must contain at least one, and only one, write with *VOLUME* as one of the data array values. For example:

```
write(0xE2, [0x04, VOLUME])
```

For the [**Set Mute**] command string, the commands must contain at least one, and only one, write with *MUTE* as one of the data array values. For example:

```
write(0xE2, [0x18, MUTE])
```

Optional Parameters

The 2.3.1.1.1 write and 2.3.1.1.2 read commands have an additional *optional* parameter called *i2c_stop* that can be omitted. If the paramter is omitted, the value is set to a default specified in the command description. For example, these are valid uses of the write command with the i2c_stop parameter:

```
write(0xE2, [0x18, 0x00], True)
write(0xE2, [0x18, 0x00], False)
write(0xE2, [0x18, 0x00], i2c_stop=False)
write(0xE2, [0x18, 0x00], i2c_stop=True)
write(0xE2, [0x18, 0x00])
```

In this last case, the *i2c_stop* parameter is not specified, so it is set to its default value, *True*.

2.4 I/O Options

The I/O Options properties are used to enable or disable the IO protocol and serial pass-through features. The IO protocol allows an application running on the host to write and read GPIO pins, read the analog input, and perform I2C transactions. The serial pass-through feature allows an application running on the host to read and write serial data via the CP2615 serial pins.

Table 2.4. I/O Configuration Options

Property	Description	Values
Enable IO Protocol	Enable the CP2615 IO protocol. If this is enabled, then a host application can access GPIO pins and other IO features.	• No • Yes
Enable Serial Protocol	Enables serial pass-through from the application to/from the CP2615 serial port.	• No • Yes
I/O Protocol Option ID	This value is exposed via the IO protocol available to a host application. If there is no host application, or the host application does not use IO protocol, then this value has no meaning.	16-bit hexadecimal

2.5 GPIO

There are two groups of GPIO signals, GPIO.0-7 and GPIO.8-15. For GPIO.0-7, each pin may be configured for an I/O mode (in, out, push-pull, open-drain, etc) and set to either a general purpose I/O or an alternate function. The alternate function can be chosen from a list of several possible functions. Please refer to the data sheet for a description of the alternate functions.

The GPIO.8-15 pins are similar to GPIO.0-7 in that each pin can be configured for an I/O mode or alternate function, except that the alternate function is fixed per pin. For example, the RTS function is limited to GPIO.11 only.

The reset value can also be configured for all GPIO pins and determines the initial value of the output pin when the CP2615 exits reset. This value is only used if the GPIO is an output.

Table 2.5. GPIO Pin Options

Property	Description	Values
GPIO.n-Pin Mode	Controls the I/O pin pad configuration	• Input • Output (push-pull) • Output (open drain)
GPIO.n-Pin Function	Chooses the alternate function for the pin, if the mode is set to something other than GPIO.	See list of alternate input and output functions in the data sheet
GPIO.n-Reset Latch	The initial logic level of the pin (if output) after CP2615 reset.	• Low • High

Table 2.6. Alternate Pin Configuration Options

Property	Description	Values
CLKOUT Divider (Hz)	If the alternate pin function CLKOUT is enabled on GPIO.12, this property is used to set the divider value for the CLKOUT frequency. Refer to the data sheet for more details about the CLKOUT signal.	integer, 1-256
UART Baud Rate	The data rate used for the UART if the external accessory (EA) serial protocol is enabled. Note that if Now Playing is enabled instead, the data rate is fixed at 115200.	• 9600 • 19200 • 38400 • 57600 • 115200
BUTTONS-Slot #	For each of the analog buttons input slots, this selects the function of the button. This is only used if GPIO.9 is configured for the alternate function BUTTONS.	• None • PLAY_PAUSE • FFWD • REW • MUTE • VOL+ • VOL- • PLAY • STOP • RECMUTE

3. Revision History

3.1 Revision 0.1

October 21st, 2016

Initial revision.

3.2 Revision 0.2

April 10th, 2017

Updated for Xpress Configurator.

3.3 Revision 0.3

April 18th, 2017

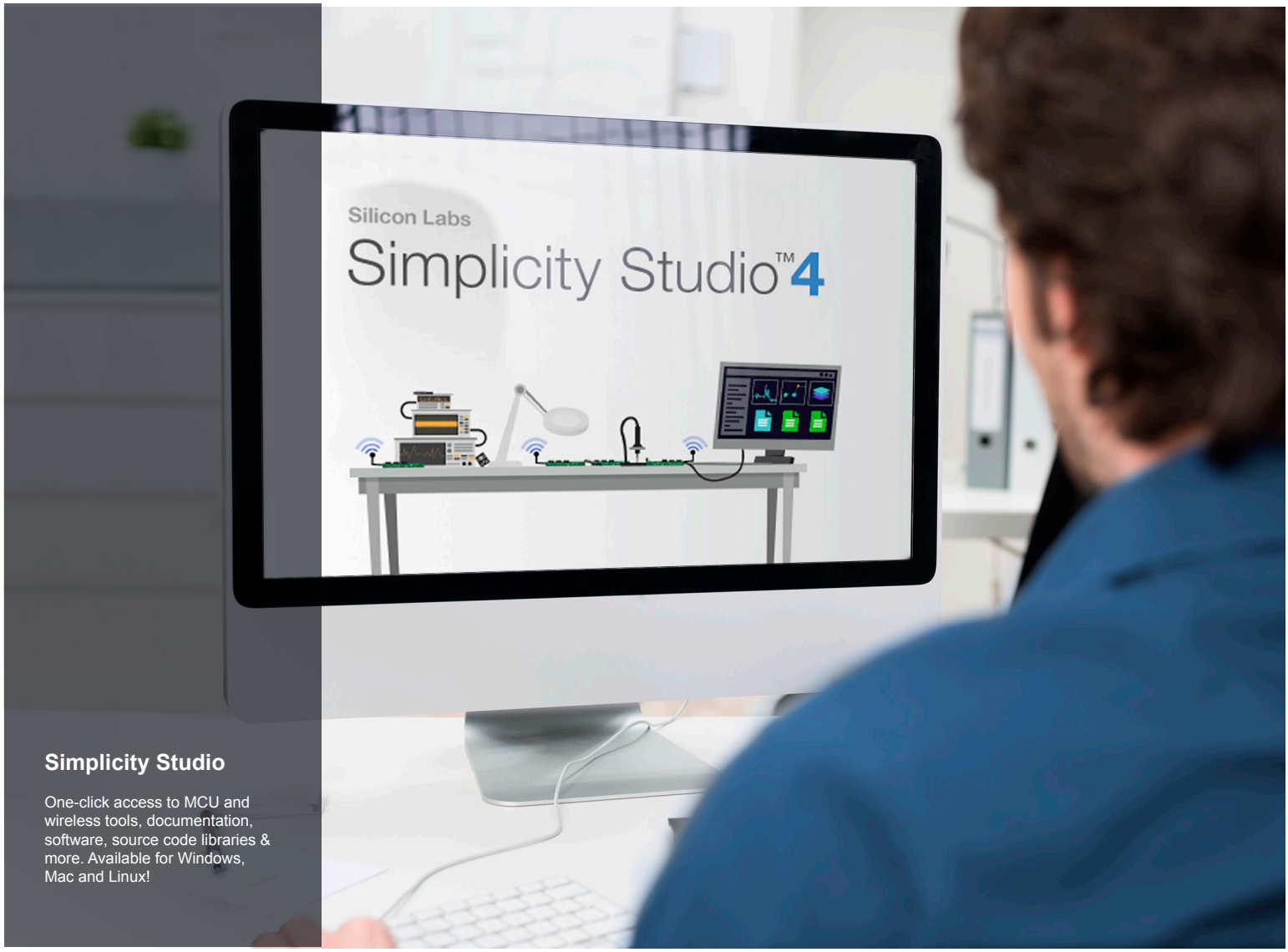
Minor fixes:

- [**Clock Sync Mode**] description listed a restriction for Asynchronous mode that doesn't exit.
- [**BUTTONS-Slot#**] listed some invalid options which were removed.
- [**Audio Interfaces**] did not list all possible options. It was missing Audio In (24 bit only) and Audio Out (24 bit only).

3.4 Revision 0.4

April 26th, 2017

Added [**Input Terminal Type**] and [**Output Terminal Type**] options to Audio Interface Options.



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com