UG173: Blue Gecko BGScript™ Developer's Guide

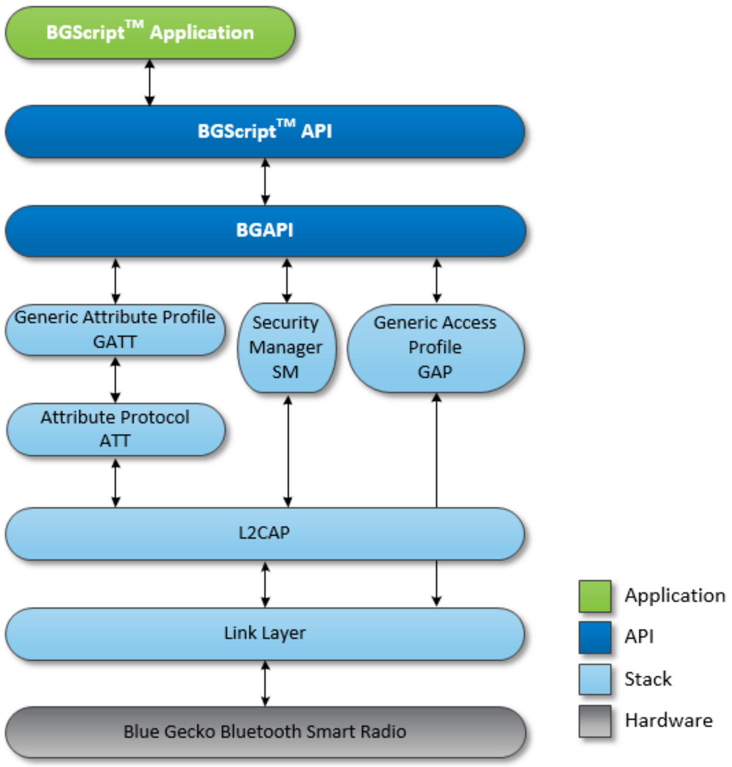
BGScript is a scripting language intended for programming of simple applications. BGScript applications can be used to automate application functionalities such as opening a connection, listening for GPIO interrupts, and even for reading and writing data via interfaces such as UART, SPI, I2C or GPIO. BGScript can also be used for simple data processing using the available BGScript arithmetic, bitwise, buffer and data comparison operations.

BGScripting allows complete applications to be implemented without the need for an external host controller (MCU), since BGScript can be executed directly on the Silicon Labs wireless module.

This user's guide describes BGScript syntax and contains useful code snippets for some of the most common functionalities used with the Silicon Labs Blue Gecko Bluetooth Smart products.

KEY FEATURES

- Create standalone applications
- Minimal programming experience needed
- Simple syntax
- Standard programming language features and operations
- Powerful functions to simplify complex tasks
- Easy access to common BGAPI features



1. BGScript Syntax

BGScript has a BASIC type of syntax. The code is executed only in response to event messages. Lines are executed in successive order. A line starts from an event definition and is finished by a ***return*** or ***end***. Each line represents a single command.

Below is an example BGScript which shows how device boot or reset is dedected and how you can enable BLE advertisements, allow connections and detect a closed connection.

Example: System start-up detection and basic BLE functionality

```
# Boot event listener - Generated when the device is started
event system_boot(major,minor,patch,build,bootloader,hw)

  # Set advertisement interval to 100ms, use all three advertisement channels
  call le_gap_set_adv_parameters(160,160,7)

  # Enable BLE advertisements and allow connections
  call le_gap_set_mode(2,2)
end

# Disconnect event listener - generated when BLE connection is closed
event le_connection_closed(reason,connection)

  # Restart BLE advertisements and allow connections
  call le_gap_set_mode(2,2)
end
```

1.1 Comments

Anything after a # character to the end of the line is ignored.

```
X=1 #comment
```

1.2 Variables and Values

1.2.1 Values

Values are always interpreted as integers (no floating-point numbers). Hexadecimal values can be expressed by putting a $ character before the value. Internally, all values are 32-bit signed integers stored in memory in little-endian format.

```
x = 12     # same as x = $0c
y = 703710 # same as y = $abcde
```

1.2.2 Variables

Variables (excluding buffers) are signed 32-bit integer containers, stored in little-endian byte order. Variables must be defined before usage.

```
dim x
dim y
x = (2 * 2) + 1
y = x + 2
```

1.2.3 Global Variables

Variables can be defined globally using dim definition which must be used outside an event block.

```
dim j
# software timer listener
event hardware_soft_timer(handle)
    j = j + 1
    call flash_ps_save(FLASH_PS_KEY_CNT, 4, j)
end
```

1.2.4 Constant Values

Constants are signed 32-bit integers stored in little-endian byte order and they also need to be defined before use. Constants can be particularly useful because they do not take up any of the limited RAM that is available to BGScript applications. Instead, constant values are stored in flash memory as part of the application code.

```
const x = 2
```

1.3 Buffers

Buffers hold 8-bit values and can be used to prepare or parse more complex data structures. For example a buffer might be used to prepare an attribute value before writing it into the attribute database.

Similar to variables buffers need to be defined before usage. Maximum size of a buffer is 256 bytes.

```
dim data(32) # Data buffer
dim len      # Data buffer length

len = 22
data(0:len) = "BLE Advertisement data"
call le_gap_set_adv_data(0, len, data(0:len))
```

Buffers use an index notation with the following format:
`BUFFER(<expression>:<size>)`

The **< *expression* >** is used as the index of the first byte in the buffer to be accessed and **< *size* >** is used to specify how many bytes are used starting from the location defined by **< *expression* >**.
Note: Note that this **<*size*>** is not the end index position.

```
u(0:1)=$a
u(1:2)=$123
```

The following syntax could be used with the same result due to little-endian byte ordering:

```
u(0:3)=$1230a
```

When using constant numbers to initialize a buffer, only four (4) bytes may be set at a time. Longer buffers must be written in multiple parts or using a string literal as indicated in the example below:

```
u(0:4)=$32484746
u(4:1)=$33
```

Buffer index and size are optional. Default values, which are 0 for index and size defined in the variable declaration for size, are used in the place of left out values.

1.3.1 Using Buffers with Expressions

Buffers can also be used in mathematical expressions, but only a maximum of four (4) bytes are supported at a time since all numbers are treated as signed 32-bit integers in little-endian format. The following examples show valid use of buffers in expressions.

```
a = u(0:4)
a = u(2:2) + 1
u(0:4) = b
u(2:1) = b + 1
```

The following example is not valid:

```
if u(0:5) = "FGH23"
    # do something
end if
```

This is because the mathematical equality operator ("=") interprets both sides as numerical values and in BGScript numbers are always 4 bytes (32 bits). This means you can only compare (with "=") buffer segments which are exactly four (4) bytes long. If you need to compare values which are not four (4) bytes in length you must use the ***memcmp*** function, which is described later in this document.

```
if u(1:4) = "GH23"
   # do something
end if
```

1.3.2 Strings

Buffers can be initialized using literal string constants. Using this method more than four (4) bytes at a time may be assigned.

```
u(0:6) = "BGM111"
```

Literal strings support C-style escape sequences, so the following example will do the same as the above example:

```
u(0:5) = "\x46\x47\x48\x32\x33"
```

1.3.3 Constant Strings

Constant strings must be defined before use. Maximum size of a constant string depends on application and stack usage. For standard applications a safe size is around 64 bytes.

```
const str() = "test string"
```

Constant strings can be used in place of buffers. Note that in the following example the index and the size of the buffer are left as default values.

```
call endpoint_send(11, str(:))
```

1.4 Expressions

Expressions are given using infix notation.

```
x=(1+2)*(3+1)
```

Table 1.1. Supported Mathematical Operators

Operation	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Less than	<
Is less than or equal to	<=
Is greater than	>
Is greater than or equal to	>=
Is equal to	=
Is not equal to	!=
Parentheses	( )

Note: Currently there is no support for modulo or power operators.

Table 1.2. Supported Bitwise Operators

Operation	Symbol
AND	&
OR	|
XOR	^
Shift left	<<
Shift right	>>

Table 1.3. Supported Logical Operators

Operation	Symbol
AND	&&
OR	||

1.5 Commands

1.5.1 event <event_name> (<event_parameters>)

A code block defined between ***event*** and ***end*** keywords is an event listener and will be run in response to a specific event. BGscript allows implementing multiple listeners for a single event. Each event listener will be executed in the order in which they appear in BGScript source code. Execution will stop when reaching the ***end*** keyword of the last event listener or the ***return*** keyword in any event listener. This event listeners chaining makes possible to implement common script files (helper libraries) for generally used operations.

BGScript VM (Virtual Machine) queues each event generated by the API and executes them in FIFO order atomically (one at a time and all the way through to completion or early termination).

```
# Boot event listener - Generated when the device is started
event system_boot(major, minor, patch, build, bootloader, hw)

    # Read GPIO status (Button PB1 on WSTK)
    call hardware_read_gpio(5,$80)(r,data)

    # check GPIO status
    if(data)
        # Start normal BLE advertisements and enable connections
        call le_gap_set_mode(2,2)
    else
        # PB1 button was pressed during boot
        # Do nothing
    end if
end
```

1.5.2 if <expression> [else] end if

Conditions can be tested with the ***if*** command. Commands between ***if*** and ***end if*** will be executed if the ***<expression>*** is true.

```
if x<2

    x=2
    y=y+1
end if
```

If ***else*** is used, then if the *<expression>* is true, commands between ***then*** and ***else*** will be executed. If the *<expression>* is false, commands between ***else*** and ***end if*** will be executed.

```
if x<2
    x=2
    y=y+1
else
    y=y-1
end if
```

Note: BGScript uses C language operator precedence. This means that bitwise "&" and "|" operators have lower precedence than the comparie operator. This means that comparisons are handled first if present in the same expression. This is important to know when creating more complex conditional statements. It is a good idea to include explicit parentheses around expressions which need to be evaluated first.

```
if $0f & $f0 = $f0
      # will match because ($f0 = $f0) is true, and then ($0f & true) is true
end if

if ($0f & $f0) = $f0
     # will NOT match because ($0f & $f0) is $00, and $00 != $f0
end if
```

1.5.3 while <expression> end while

Loops can be constructed using the ***while*** command. Command lines between ***while*** and ***end while*** will be executed when the ***<expression>*** is true.

```
a=0
while a<10
     a=a+1
end while
```

1.5.4 call <command name>(<command parameters>..)[(response parameters)]

The ***call*** command is used to execute BGAPI commands and receive command responses. Command parameters can be given as expressions while response parameters are variable names into which response values will be stored. Response parentheses and parameters can be omitted if the response is not needed by your application.
Note: All response variables must be declared before use.

```
# Generated when GATT characteristic client configuration value is changed
 event gatt_server_characteristic_status(connection,characteristic,status_flags,client_config_flags)

  # Check if indications have been enabled for HTM temperature measurement and the timer is not running
  if (characteristic = xgatt_temperature_celsius) && (status_flags = 1) && (client_config_flags = 2) && (timer
= 0) then
    # Indications were enabled - set software timer to tick each second
    call hardware_set_soft_timer(4096,0,0)
    timer = 1
  end if
end
```

The ***call*** command can also be used to execute user-defined procedures (functions). The syntax in this case is similar to executing a BGAPI command, except that return values are not supported.

1.5.5 let <variable> = <expression>

The ***let*** command is an optional command which can be used to assign an expression to a variable.

```
let a=1
let b=a+2
```

1.5.6 return

This command causes a return from an event or a procedure.

```
# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)

  # Read GPIO status (Button PB1)
  call hardware_read_gpio(5,$80)(r,data)

  # check GPIO status
  if(data)
    # Button was not pressed - do not start advertisement
    return
  else
    # PB1 button was pressed during boot
    call le_gap_set_mode(2,2)
  end if
end
```

1.5.7 float (mantissa,exponent)

The float command can be used to change a given mantissa and exponent in to a 32-bi base-10 IEEE 11073 SFLOAT value. Conversion is done using the following algorithm:

	Exponent	Mantissa
Length	8-bit	24-bit
Type	signed integer	signed integer

The list below defines reserved special purpose values

- `NaN` (Not a Number)
  - exponent `0`
  - mantissa `0x007FFFFF`
- `NRes` (Not at this resolution)
  - exponent `0`
  - mantissa `0x00800000`
- Positive infinity
  - exponent `0`
  - mantissa `0x007FFFFE`
- Negative infinity
  - exponent `0`
  - mantissa `0x00800002`
- Reserved for future use
  - exponent `0`
  - mantissa `0x00800001`

1.5.8 memcpy(destination,source,length)

The ***memcpy*** command copies bytes from the defined source buffer to the defined destination buffer. The destination and source should not overlap. Note that the buffer index notation only uses the start byte index and should not include the size portion.

Example: `dst(start)` instead of `dst(start:size)`.

```
dim dst(3) dim src(4)
memcpy(dst(0),src(1),3)
```

1.5.9 memcmp(buffer1,buffer2,length)

The memcmp function compares buffer1 and buffer2 , for the length defined with length . The function returns the value **1** if the data is identical.

```
dim x(3) dim y(4)
if memcmp(x(0),y(1),3)
    #do something
end if
```

1.5.10 memset(buffer , value , length)

This command fills the defined buffer with the data defined in value for the length defined.

```
dim dst(4)
memset(dst(0), $30, 4)
```

1.6 Procedures

BGScript supports procedures which can be used to implement subroutines. Procedures differ from functions used in other programming languages since they do not return a value and cannot be used in expressions.

Procedures are called using the ***call*** command just like other BGScript commands.

Procedures are defined by the command ***procedure*** as shown below. Parameters are defined inside parentheses the same way as in event definition. Buffers are defined as the last parameter and require a pair of empty parentheses.

Example: Printing the MAC Address of the Wi-Fi Module

```
dim n,j
dim addr(6)

procedure print_nibble(nibble)
      n=nibble
      if n<$a
            n=n+$30
      else
            n=n+$37
      end if
      call endpoint_send(0,1,n)
end

procedure print_hex(hex)
      call print_nibble(hex/16)
      call print_nibble(hex&$f)
end

procedure print_mac(len,mac())
      j=0
      while j<len
            call print_hex(mac(j:1)) j=j+1
            if j<6
                 call endpoint_send(0,1,":")
            end if
      end while
end

# Boot event listener
event system_boot(major, minor, patch, build, bootloader, hw)
  # Read MAC address
  call system_get_bt_address()(addr(0:6))
  # Print address
  call print_mac(6,addr(0:6))
end
```

1.7 Using multiple BGscript files

1.7.1 Import

The ***import*** directive allows you to include other BGscript files.

main.bgs

```
import "other.bgs"

event system_boot(major, minor, patch, build, bootloader, hw)
    # device module has booted
end
```

1.7.2 Export

By default all code and data are local to each BGscript file.
The ***export*** directive allows use of variables and procedures in another script file, which imports the script file where variables are defined.

other.bgs

```
export dim hex(16)
export procedure init_hex()
hex(0:16) = "0123456789ABCDEF"
end
```

main.bgs

```
import "other.bgs"
event system_boot(major, minor, patch, build, bootloader, hw)
    call init_hex()
end
```

2. BGScript Limitations

2.1 32-bit Resolution

All values used in BGScript must fit into 32 bits. This affects e.g. definition of very long timer intervals.

2.2 Performance

BGScript has limited performance (commands/operations per time unit), which might prevent some applications from being implemented using BGScript.

BGScript can typically execute commands/operations in the order of thousands of commands per second.

3. BGScript Examples

This section contains examples on how to perform various actions with BGScript.

3.1 Basic examples

This section contains some very basic BGScript examples.

3.1.1 Catching System Startup

This example shows how to catch a system start-up. This event is the entry point to all BGScript code execution and can be compared to `main()` function in C.

Example: Catching system start-up

```
# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)
end
```

3.1.2 Performing a System Reset

This example shows how to perform a system reset.

Example: System reset

```
# Something went wrong and system needs to be reset
call system_reset(0)
```

This example shows how to reset the device into Device Firmare Update (DFU) mode.

Example: DFU reset

```
# Reset the device into DFU mode
call system_reset(1)
```

3.2 Bluetooth Smart

This section shows simple BGScript code examples for handling Bluetooth Smart related events.

3.2.1 Advertising

Bluetooth Smart advertisement in used to make the local device visible to other devices and it can also be used to allow connections. Bluetooth Smart also allows up to 30 bytes of data to be broadcasted in the advertisement packets and the end user can decide what the data is.

Example: Enabling advertisement and connections

```
# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)

    # Set advertisement interval to 100ms, use all three ADV channels
    call le_gap_set_adv_parameters(160,160,7)

    # Start Bluetooth LE advertisements and enable connections
    call le_gap_set_mode(2,2)
end
```

Example: Enabling advertisement with custom advertisement data

```
# Advertisement data
dim data(4)

# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)

    # build advertisement data
    data(0:1) = $01
    data(1:1) = $02
    data(2:1) = $03
    data(2:1) = $04

    # Set advertisement interval to 1000ms, use all three ADV channels
    call le_gap_set_adv_parameters(1600,1600,7)

    # Start Bluetooth LE advertisements and enable connections
    call le_gap_set_adv_data(0, 4, data(0:4))

    # Start Bluetooth LE advertisements and enable connections.
    # ADV mode 4 must be used with custom data.
    call le_gap_set_mode(4,2)
end
```

3.2.2 Scanning

Bluetooth Smart scanning the the operation needed to discover other Bluetooth devices and/or initiate connections. Scanning can also be used to receive the broadcast data from devices that are advertising.

Example: Enabling scanning

```
# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)

    # Set scan parameters Interval: 100ms, window: 10ms, passive scanning
    call le_gap_set_scan_parameters(160, 16, 1)

    # Start a generic BLE discovery
    call le_gap_discover(1)
end

# Catching scan response events - generated when an adevertisement packet is received
event le_gap_scan_response(rssi, packet_type, address, address_type, bonding, data_len, data_data)

    # Try to connect if the remote device is connectable
    if (packet type = 0) then
        call le_gap_open(address, address_type)
    end if
end
```

3.2.3 Bluetooth Connection Events

This example shows how to catch a Bluetooth Smart connection event, which is generated when the local device is advertising, and a remote device connects to it.

Exampe: Catching a BLE connection events

```
# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)
    # Start normal BLE advertisements and enable connections
    call le_gap_set_mode(2,2)
end

# Connection event listener - generated when BLE connection is opened
event event le_connection_opened(address, address_type, master, connection, bonding)
    #Do Nothing
end

# Disconnect event listener - generated when BLE connection is closed
event le_connection_closed(reason,connection)
    # Restart advertisement
    call le_gap_set_mode(2,2)
end
```

3.2.4 Security

The examples here show how to enable Bluetooth Smart Security

Example: Enabling Just Works Security mode and bonding

```
# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)

    # Enable Just Works security mode. No MITM, no input/output
    call sm_configure(0,3)

    # Enable bonding
    call sm__set_bondable_mode(1)

    # Start Bluetooth LE advertisements and enable connections.
    call le_gap_set_mode(2,2)
end

# New bonding received
event sm_bonded(connection, bonding)

    # Check if bonding handle was created
    if (bonding = $ff) then
        # Bonding handle created
    end if
end
```

3.3 GATT

This section shows simple BGScript code examples for handling GATT data base events and data transactions.

3.3.1 Read and Write

GATT server's read and write operations can be used to write values to the local GATT database, which the remote device can then access.

Example: Writing a value to a local GATT database using GATT server

```
# Software timer event - generated when software timer runs out
event hardware_soft_timer(handle)

    # Read temperature from the RHT sensor
    # After using this procedure, exported variable "data" will carry the temperature value and "len" length o
f the value
    call read_i2c()

    # Build HTM service's temperature reading characteristic
    # Set flags for Celsius temperature
    tmp(0:1)=0
    # Convert the value to float
    tmp(1:4)=float(data*1757/65536-469, -1)

    # Write attribute to local GATT data base Temperature Measurement attribute
        call gatt_server_write_attribute_value(xgatt_temperature_celsius,0,5,tmp(0:5))(result)
end
```

3.3.2 Indicate and Notify

Bluetooth Smart advertisement in used to make the local device visible to other devices and it can also be used to allow connections. Bluetooth Smart also allows up to 30 bytes of data to be broadcasted in the advertisement packets and the end user can decide what the data is.

Example: Detecting when indications or notifications are enabled or disabled

```
# Generated when GATT characteristic client configuration value is changed
event gatt_server_characteristic_status(connection,characteristic,status_flags,client_config_flags)

    # Check if indications have been enabled for "xgatt_temp" characteristic
    if (characteristic = xgatt_temp) && (status_flags = 1) && (client_config_flags = 2) then
    # Indications were enabled - do something f.ex start sensor readings
    end if

    # Check if indications have been disabled for "xgatt_temp" characteristic
    if (characteristic = xgatt_temp) && (status_flags = 1) && (client_config_flags = 0) then
    # Indications were disabled - do something f.ex stop sensor readings
    end if

    # Check if notifications have been enabled for "xgatt_hr" characteristic
    if (characteristic = xgatt_hr) && (status_flags = 1) && (client_config_flags = 1) then
    # Notifications were enabled - do something f.ex start sensor readings
    end if

    # Check if notifications have been disabled for HTM temperature measurement
    if (characteristic = xgatt_hr) && (status_flags = 1) && (client_config_flags = 0) then
    # Notifications were disabled - do something f.ex stop sensor readings
    end if
end
```

Example: Indicating a value to a remote device

```
# Software timer event - generated when software timer runs out
event hardware_soft_timer(handle)

    # Read temperature from the RHT sensor
    call read_i2c()

    # Build HTM service's temperature reading characteristic
    # Set flags for Celsius temperature
    tmp(0:1)=0
    # Convert the value to float
    tmp(1:4)=float(data*1757/65536-469, -1)

    # Write attribute to local GATT data base Temperature Measurement attribute
    call gatt_server_write_attribute_value(xgatt_temperature_celsius,0,5,tmp(0:5))

    # Send indication to all "listening" clients
    # 0xFF as connection ID will send indications to all connections
    call gatt_server_send_characteristic_notification($ff, xgatt_temp,5,tmp(0:5))
end
```

3.4 Timers

This section describes how to use timers with BGScript.

3.4.1 Continous and Single Interrupt Timers

The examples below show how to generate continuous and single timer interrupts.

Example: Generating and catching timer interrupts

```
# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)
    # Start a continuous software timer that generates events every 1000ms
    # 4096 = 1000ms, 0 = handle, 0 = continuous timer
    call hardware_set_soft_timer(4096,0,0)

    # Start a single shot timer that generates an event after 2000msms
    # 4096 = 1000ms, 1 = handle, 1 = continuous timer
    call hardware_set_soft_timer(8128,1,1)
end

# Timer event - generated when a timer event occurs
event hardware_soft_timer(handle)

    # Continous timer event
    if (handle = 0) then
        # Do something
    end

    # Single shot timer event
    if (handle = 1) then
        # Do something else
    end
end
```

3.5 Endpoints

This section contains examples on how to utilize endpoints using BGScript.

3.5.1 UART Endpoint

An UART endpoint can operate in two different modes: streaming or BGAPI.

In streaming mode any incoming UART data is transparently routed to another endpoint. A typical use case for this is sending and receiving TCP /IP data through UART.

Example: Enabling UART interfaces in hardware.xml file to allow BGScript access

```
<hardware>
   ...
   <!-- UART configuration -->
   <!-- Settings: @115200bps, no RTS/CTS and BGAPI serial protocol is disabled -->
   <uart index="1" baud="115200" flowcontrol="false" bgapi="false"/>
   ...
</hardware>
```

Example: Writing to a UART endpoint

```
# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)

   # Write to UART1 endpoint
   call endpoint_send(2, 6, "boot\r\n")
end
```

3.6 PS Store

This section contains examples on how to use PS keys using BGScript.

PS Store is a module internal nonvolatile memory (NVM) data storage structure, with size of 4kB and it is duplicated to avoid data loss. The structure of storage is key – value pairs, where key is unique identifier to data.

3.6.1 Reading and Writing PS Keys

The examples below show how to access PS keys.

Example: Write and read the value of a single PS key

```
dim result
dim length
dim data(6)

# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)
    # Save key 0
    call flash_ps_save($4000, 6, "BGM111")
    # Read key 0
    call flash_ps_load($4000)(result,length, data(0:length))
end
```

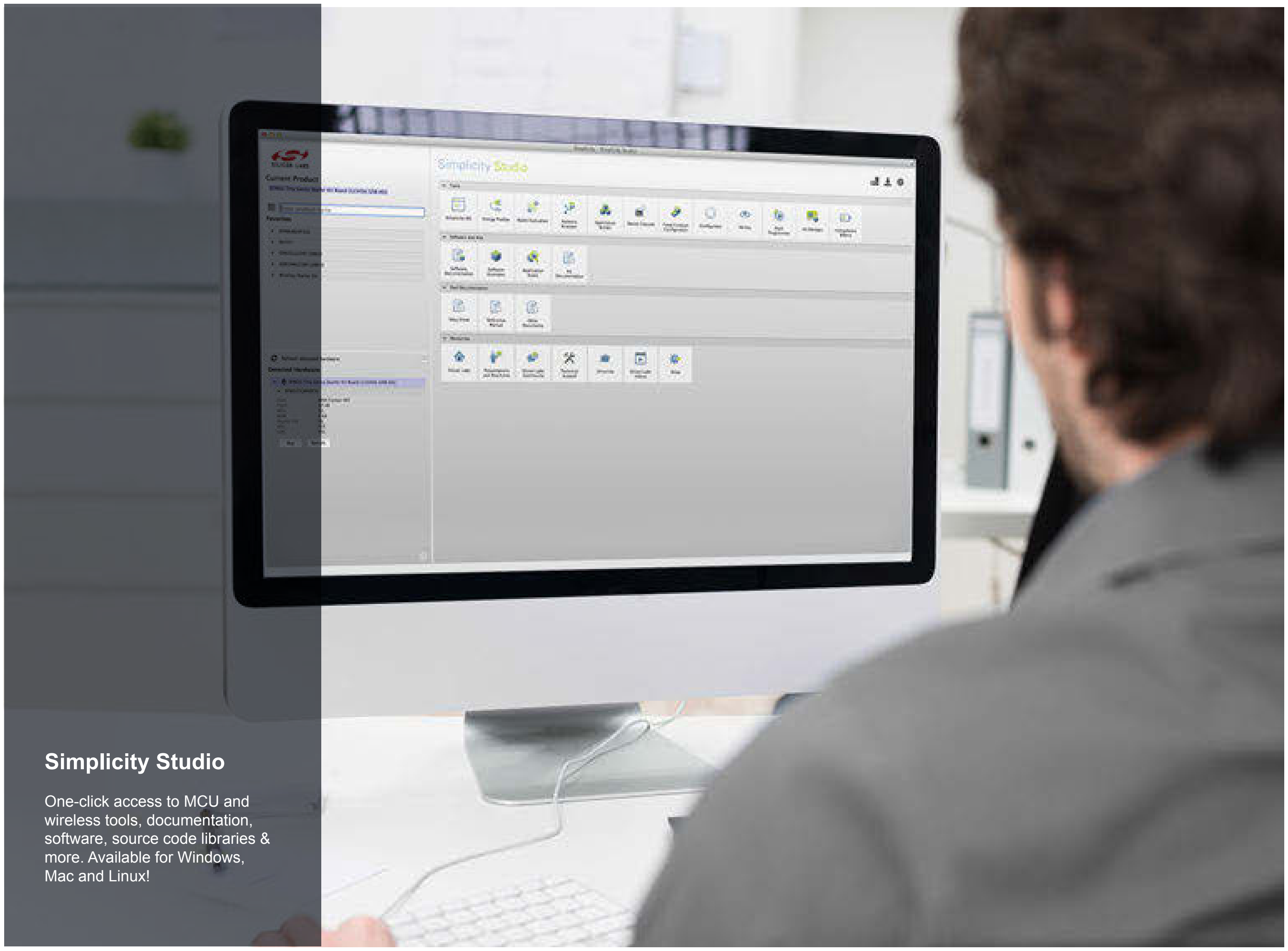
Note: The application can only use PS-keys with IDs from 0x4000 to 0x407F.

3.6.2 Deleting PS Keys

This example shows how to delete a single PS key.

Example: Deleting a single PS key

```
# Boot event listener - Generated when the module is started
event system_boot(major,minor,patch,build,bootloader,hw)
    # Erase key 0
    call flash_ps_erase(0)
end
```



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