

## SMARTBASIC FOR BLUETOOTH LOW ENERGY MODULES

---

### Introduction

*smartBASIC* is an implementation of a structured BASIC programming language optimised for use on low-cost embedded systems with limited memory by being highly efficient in terms of memory usage.

The BASIC programming language has been around for over 40 years in many variants and platforms and a good background and introduction is available at <http://en.wikipedia.org/wiki/BASIC>.

### What Are the Reasons for writing *smart* BASIC Applications?

*smartBASIC* was designed to make BLE (Bluetooth Low Energy) development quicker and simpler, vastly cutting down time to market. The following are three good reasons for writing applications in *smartBASIC*:

- Since the Laird Technologies BL600 module can auto run the application every time it powers up, you can implement a complete design within the module.
- Many use cases would combine the BL600 with a sensor device. *smartBASIC* allows sensor data to be manipulated using arithmetic and string processing functions allowing intelligent use of the Bluetooth Low Energy radio.
- If you already have a product with a wired communications link, such as a modem, you can write a *smartBASIC* application for one of our wireless modules that copies the interface for your wired module. This provides a fast way for you to upgrade your product range with the minimum number of changes to its existing firmware.

### Modes of Operation

The BL600 BLE module running *smartBASIC* has two different modes of operation:

- **Interactive Mode** – In Interactive mode, commands are sent via the UART and are executed immediately, analogous to the behavior of a modem using AT commands. Interactive mode can be used by a host processor or a PC to directly configure and control the module. It is also used to manage the download and storage of BASIC applications in the flash file system that is subsequently used in Run-time mode.
- **Run-time Mode** – In Run-time mode, the module reads pre-compiled BASIC applications directly from program memory and executed in-situ. The ability to interpret the application from flash ensures that the maximum amount of RAM memory is available to the user application for data variables.

### Autorun

By default, the module starts up in Interactive mode and checks to see if an application called **\$autorun\$** exists in the file system. If present, it automatically runs unless the **nAutoRUN** input pin is used to signal that the module should skip that step.

If the autorun application exits, which in virtually all cases is not the case, then the module returns to Interactive mode. It is possible to write autorun applications that continue to run to control the module's behaviour until power-down, which provides a *complete embedded application*.

### smart BASIC Essentials

Being a structured programming language, *smartBASIC* offers typical modern constructs such as subroutines, functions, **while**, **if**, and **for** loops. Applications written in *smartBasic* are event-driven rather than the sequential processing seen in early versions of BASIC.

A typical *smartBASIC* application source code consists of the following:

- Variable declarations and initialisations
- Subroutine definitions
- Event handler routines
- Startup code

The source code ends with **WAITEVENT**, a final statement which never returns. Once the run-time engine reaches the **WAITEVENT** statement, it waits for events to happen and, when they do, it calls the appropriate handlers (written by the user) to service them.

The core language, which is common throughout all *smartBASIC* implementations, provides the standard functionality of any program, such as:

- Arithmetic functions
- Binary operators
- Conditionals
- Looping
- Functions and subroutines
- String processing functions
- Arrays (single dimension only)
- I/O functions (GPIO, ADCs, I2C, SPI and UART)
- Memory management
- Event handling

In addition *smartBASIC* adds Bluetooth Low Energy Extensions to control the wireless connectivity of the BL600:

- Advertising
- Connecting
- Security – encryption and authentication
- Power management
- Wireless status

*smartBASIC* applications can be written in any standard text editor such as textpad or Notepad++ and do not require a complex or costly development environment. If you prefer syntax colour highlighting, then treating the source as C or BASIC will display the code with colour highlights. Applications are transferred to the module using a simple free terminal program called UWTerminal (available for download from [www.lairdtech.com/wireless](http://www.lairdtech.com/wireless)). Customers are able to develop their own utilities to download precompiled applications as the process involves the use of standard AT commands.

## Sample: SmartBASIC BLE Application

```

'// Simple program that checks a digital io every 5000ms and begins to advertise if high

'// Declare variables

DIM gpiostate           '// declare an integer variable called gpiostate
DIM rc                  '// declare an integer variable called rc, rc
                        '// stands for result code

'// User Function

FUNCTION handlerTimer0()           '// this handler function is called when Timer 0
                                    '// expires

    print "Timer 0 has expired, checking GPIO\n"           '// prints to the UART
    gpiostate = gpioread(15)                               '// reads the value of the GPIO which now becomes
                                                            '// the variable "number"

    if gpiostate == 1 then                                       '// if the GPIO is high then

        print "pin value is high, begin advert\n"           '// prints to the UART
        rc = bleadvertstart(25, 5000, 0)                   '// call the BLE advertising start function where
                                                            '// 25 is the interval between advertisements in
                                                            '// ms and 5000 is the time in ms after which the
                                                            '// module stops advertising

    endif
    TIMERSTART(0,5000,1)                                       '// start a timer where 0 is the timer number,
                                                            '// 5000 is the duration in ms and 1 indicates a
                                                            '// recurring timer

    PRINT "Timer 0 has now restarted\n"                       '// prints to the UART
ENDFUNC 1                                                       '// non zero return will reprocess waitevent, 0
                                                            '// will process next command after waitevent

'// Main body executed on program start

ONEVENT EVTMR0 CALL handlerTimer0                               '// when timer 0 expires run the timer0 handler
                                                            '// function

TIMERSTART(0,1000,0)                                           '// start a timer where 0 is the timer number,
                                                            '// 1000 is the duration in ms, and 0 indicates
                                                            '// a non-recurring timer

PRINT "Beginning initial timer0\n"                             '// prints to the UART

WAITEVENT                                                       '// wait in low power for an event to happen

PRINT "program has ended"                                       '// prints to the UART, this line should never
                                                            '// happen

```

## APPENDIX: GLOSSARY OF BASIC PROGRAMMING TERMS

<b>Arithmetic</b>	SmartBASIC processes arithmetic through inline statements. The traditional operators ( + , - , / , * ) may be used in a mathematical operation. In this way, algebraic operations may be executed and saved to a variable.
<b>Variable Types</b>	There are two type of variables:- INTEGER and STRING. INTEGER variables are signed 32 bit values. STRINGS can be as long as 65500 bytes, but depending on the platform and memory constraints, they are limited to a much smaller length.
<b>Arrays (single dimension)</b>	To store a collection of numbers as a group, smartBASIC uses a single-dimension array. Arrays are useful for keeping related data stored within one variable.
<b>BLE-specific syntax</b>	Laird's smartBASIC has been designed to provide specific syntax to expose BLE functionality. Special commands have been created to make the device discoverable, to advertise, to connect, to pair and other BLE actions: <ul style="list-style-type: none"> <li>▪ bleadvertstart(interval, duration, recurring?[1 = yes, 0=no])</li> </ul>
<b>Event Handling</b>	Because BLE is designed to save power, the device is typically dormant until action is required. In a smartBASIC application, this is the <b>WAITEVENT</b> stage. The command <b>ONEVENT</b> is used to declare an event that will occur as well as the function that should be run as a result. This structure means that the program can wait for events and react accordingly. Importantly, while waiting events the BL600 can be in the lowest possible power mode.
<b>Function</b>	A line of code (part of a source code within a larger computer program) that is often used more than once. <i>Calling</i> a function saves programming time (and space). Unlike a <b>subroutine</b> , a function returns a value. <p><b>Example:</b></p> <p>For a program that calculates sales tax, a function asks for a subtotal, takes that number and multiplies it by 1.07 (for a sales tax of 7%; the '1' is because the program is adding onto the subtotal to make a total, rather than finding the sales tax itself). The function returns the new total back to the program.</p>
<b>Subroutine</b>	A line of code (part of a source code within a larger computer program) that is reused. <i>Calling</i> a subroutine saves time (and space). Unlike a <b>function</b> a subroutine does not return a value.
<b>I/O Commands</b>	The status of GPIO pins, ADCs, I2Cs, and other on-board systems may be read and used to produce reactions within a SmartBASIC program. Commands include: <ul style="list-style-type: none"> <li>▪ gpioread(pin) – read the state (low or high) of a GPIO pin.</li> </ul>
<b>PRINT</b>	BASIC command that writes to the device's UART. <p><b>Example:</b></p> <pre>PRINT "[whatever you want printed to the UART]"</pre>

<p><b>Looping</b></p>	<p>Looping involves a series of statements which must be repeated several times. Using loops reduces the number of program lines and prevents you from having to re-write the same segment code multiple times.</p> <ul style="list-style-type: none"> <li>▪ For</li> <li>▪ While</li> </ul> <p><b>FOR Loop</b> In a BASIC <b>FOR</b> loop, a command is executed repeatedly until a condition of its execution is broken. In a FOR loop, you specify a number of iterations and perform a set of commands for each iteration.</p> <p><b>WHILE Loop</b> A <b>WHILE</b> loop is much like a <b>FOR</b> loop, except that the program does not define the number of iterations. Instead, a WHILE loop specifies that while a certain conditional statement is true, perform a set of commands. When the conditional turns up false, the WHILE loop ends.</p>
<p><b>Logic and Conditionals</b></p>	<p>In a <i>smartBASIC</i> application, conditional statements allow the program to evaluate a condition and then, based on the status of that condition, proceed in one of two different ways. This is achieved using an <b>IF</b> statement, which will evaluate a conditional statement and execute one set of commands if the statement is true. If the conditional is false, an <b>ELSE</b> statement directs the program to run another set of commands. <b>IF</b> statements may be nested within IF statements as well, allowing for complex, branched behaviour dependent upon many conditions.</p>
<p><b>Strings and String Processing</b></p>	<p>Strings may be saved to variables in <i>smartBasic</i>, and furthermore may be manipulated and controlled by different commands. String-manipulating commands include:</p> <ul style="list-style-type: none"> <li>▪ LEFT\$(x,y) – Grab part of string X, of length Y.</li> <li>▪ RIGHT\$(x,y) – Grab part of string X starting from the end backward, of length Y.</li> </ul> <p>Strings may also be added to each other, i.e. A\$+B\$. The result is the second string appended to the end of the first.</p>
<p><b>Variable Declarations</b></p>	<p>Used to manually define the variable name and type (such as Number, String, or Date).</p> <p>Note: If a variable name does NOT end with a \$ then it will be typed as an integer unless an AS STRING is used to override it. Likewise a name ending with a \$ will default to a STRING unless overridden with AS INTEGER</p> <p><b>Example:</b></p> <pre> DIM gpiostate    \//declare an integer variable called gpiostate DIM rc           \//declare an integer variable called rc, rc                   being Result Code DIM myname\$ AS STRING \//declare a string variable                 </pre>