AN216655

WICED HCI Test System

Associated Part Family: CYW20706, CYW20719, CYW20735

WICED™ Studio 4

To get the latest version of this application note, please visit www.cypress.com/AN216655

This document provides information on the WICED HCI Test System that can be used to test Cypress CYW207XX WICED Bluetooth devices.

Contents

1 Introduction

This document applies only to CYW207XX-based devices running embedded WICED Studio Bluetooth applications. This document will use the CYW20706A2 device for all device-specific references.

2 IoT Resources

Cypress provides a wealth of data at http://www.cypress.com/internet-things-iot to help you to select the right IoT device for your design, and quickly and effectively integrate the device into your design. Cypress provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates. Customers can acquire technical documentation and software from the Cypress Support Community website (http://community.cypress.com/).

3 Hardware and Software Requirements

The following hardware and software components are required to support the testing described in this document:

- One CYW20706-based device (including antennas)
- One third-party Bluetooth device or a second CYW20706-based device (including antennas)
- The WICED Studio Development System (which includes support for the WICED HCI Test system)
- A test computer running Windows, Linux, or Mac OS X
- A USB to Micro-USB or a USB to Mini-USB cable for connecting the test computer to a CYW20706-based device

PRELIMINARY

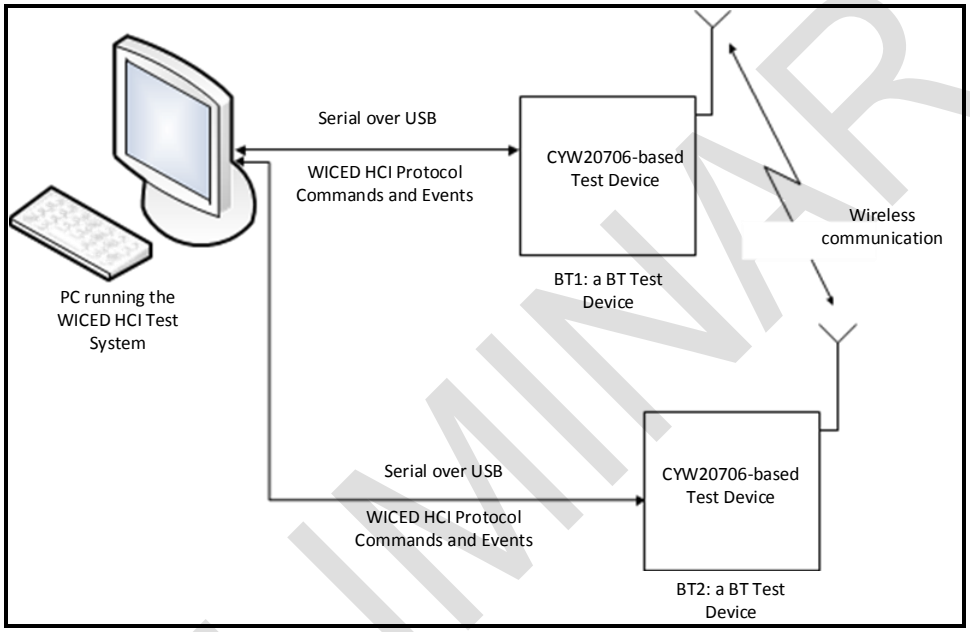
4 Introduction

The WICED HCI Test system provides an automated testing capability for CYW20706-based devices. To use the WICED HCI Test system, the CYW20706-based test device must be running an application that supports the WICED HCI protocol.

WICED Studio provides several sample applications that support the WICED HCI protocol. The source code for these applications is located in the <WICED-Studio>\<BT-Controller>\Apps directory. Developers can also create new test-device applications that support the WICED HCI protocol. For information on the WICED HCI protocol, refer to Sample HCI UART Control Protocol [1].

Figure 1 shows a typical automated testing setup where two CYW20706 devices are used.

Figure 1. Automated Testing Setup - Two CYW20706-Based Devices



In a typical test scenario, the PC running the WICED HCI Test system (see Figure 1) runs a Perl or Python script that instructs the application running on the CYW20706 device to:

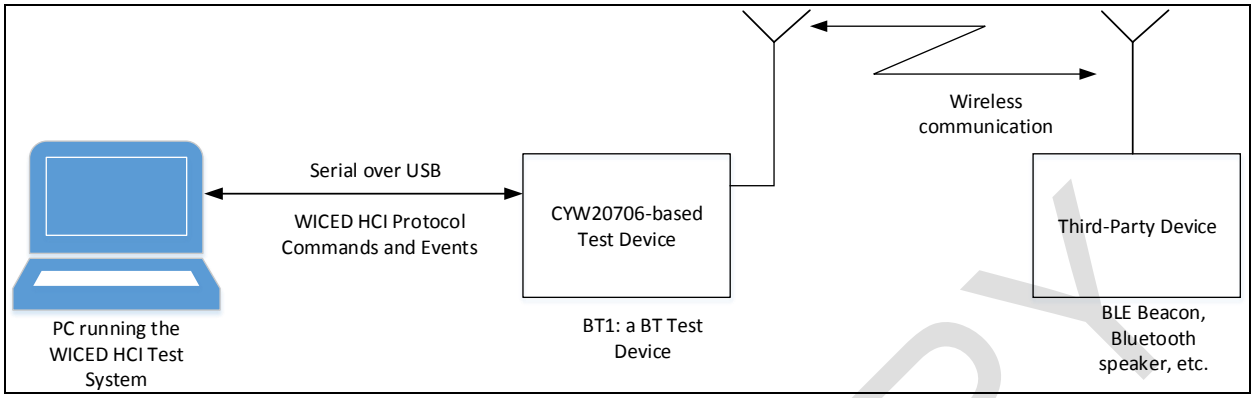
- Perform one or more Bluetooth operations.
- Verify that the application sends back the appropriate events with appropriate result codes.

The WICED HCI Test system can test all Bluetooth functionality exposed by an application running on the CYW20706 device. Testing can be performed on the Bluetooth test device (BT1) attached to the PC and on another Bluetooth test device (BT2) within the wireless range of BT1.

Figure 2 shows another possible test setup where testing is performed against a well-known third-party device. For example, if there is a known Bluetooth Low Energy (BLE) beacon in the area, a test script can instruct an application running on BT1 to perform a BLE scan and verify that the application reports the known beacon in the scan result. Similarly, if there is a Bluetooth speaker in the vicinity of a CYW20706 device, then the test script can send a command to BT1 to establish an audio or remote control connection and verify that a connection has been established successfully.

PRELIMINARY

Figure 2. Automated Testing Setup – A CYW20706-Based Device and a Third-Party Device



5 WICED HCI Test System Description

The core elements of the WICED HCI Test system include the following items:

- The HWTALK command-line utility, which is the main engine of the WICED HCI Test system. See HWTALK Command-Line Utility for more information
- WICED HCI protocol structure (or message) definition files (SDLs) for commands and events. See Structure Definition Language Files for more information
- An example Perl script
- Several example Python scripts

The WICED HCI Test system is packaged with WICED Studio. It is located in <WICED-Studio>\test\xsdl.

Figure 3 shows the home directory of the WICED HCI Test system in WICED Studio.

Figure 3. WICED HCI Test System Location in WICED Studio

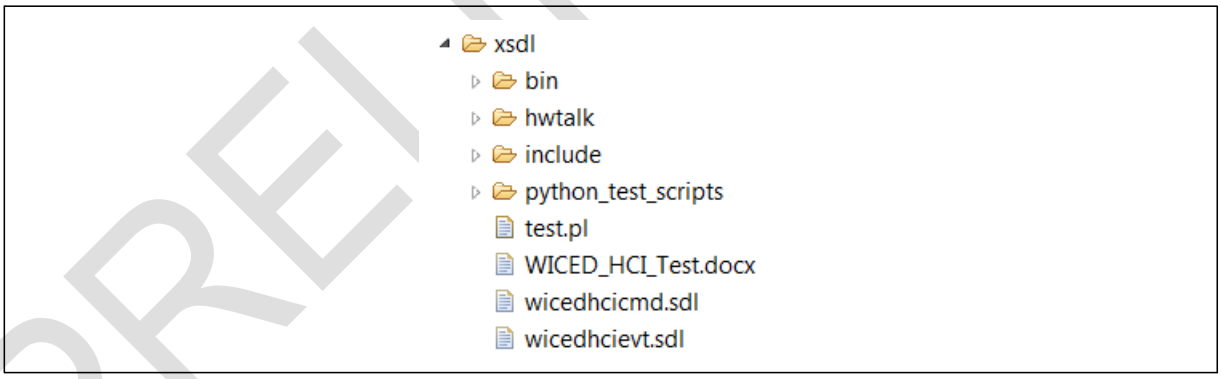


Table 1 provides definitions of the top-level WICED HCI Test system directories and files.

Table 1. File and Directory Definitions

Directory of File	Description
bin	Mac OS X and Windows binaries, and the Windows executable *hwtalk.exe*.
hwtalk	hwtalk source code
include	hwtalk include files
python_test-scripts	Python test scripts that use the hwtalk utility. This includes the *wiced_bt.py* library file, which contains a useful class definition (`wiced_bt_class`) to simplify automation.

PRELIMINARY

Directory of File	Description
test.pl	A Perl test script for demonstrating WICED HCI Test system usage
wicedhcicmd.sdl	A structure definition file for commands
wicedhcievt.sdl	A structure definition file for events

6 HWTALK Command-Line Utility

The *hwtalk.exe* command-line utility, commonly referred to as HWTALK, is the main processing engine of the WICED HCI Test system.

At execution startup (see Command-Line Parameters on page 5 for execution syntax), HWTALK does the following:

- Processes command-line arguments
- Connects to the specified COM port
- Configures its baud rate to the specified rate
- Reads the specified SDL files
- Optionally configures logging to the specified file
- Optionally enables/disables the sending of HCI and WICED traces to the BTSpy utility for decoding and display

After the HWTALK execution begins, it processes standard input (STDIN) as commands and provides standard output (STDOUT) as responses. See Commands and Responses for a description of the various commands that can be issued to HWTALK and the possible responses from HWTALK.

6.1 Command-Line Parameters

The following syntax applies to the HWTALK utility:

```
hwtalk –PORT <portid> –BAUDRATE <baud> –MSGCMDDEFS <cmdFile>
       –MSGEVTDEFS <eventFile> [–LOGTO <logfile>] [–LOGTOSPY <enable>]
```

Table 2 provides descriptions of the HWTALK command-line parameters.

Table 2. HWTALK Command-Line Parameter Descriptions

Command-Line Switch and Argument	Required or Optional	Description
`–PORT <portid>`	Required	`portid` is a standard platform-specific transport specifier. For example, COM3 on a Windows system or tty-serxxx on a Linux or Mac OS X system.
`–BAUDRATE <baud>`	Required	`baud` is the line speed of the connection. It must match the rate used in the embedded application running on the CYW20706. For example, 115,200 bps or 3,000,000 bps.
`–MSGCMDDEFS <cmdFile>`	Required	`cmdFile` is the file containing the command message structure definitions.
`–MSGEVTDEFS <eventFile>`	Required	`eventFile` is the file containing the event message structure definitions.
`–LOGTO <logfile>`	Optional	`logfile` is an optional file to which logging takes place.
`–LOGTOSPY <enable>`	Optional	`enable` is an optional parameter for enabling or disabling the sending of HCI and WICED traces to the BTSpy utility running on a Windows PC equipped with a Cypress Bluetooth device. Traces are sent by default. 1 = Enable. 0 = Disable.

6.2 Commands and Responses

The HWTALK utility processes the commands received via STDIN and provides responses via STDOUT. The following commands can be sent to HWTALK:

Send (see “Send”)

PRELIMINARY

Next (see “Next”)

Get (see “Get”)

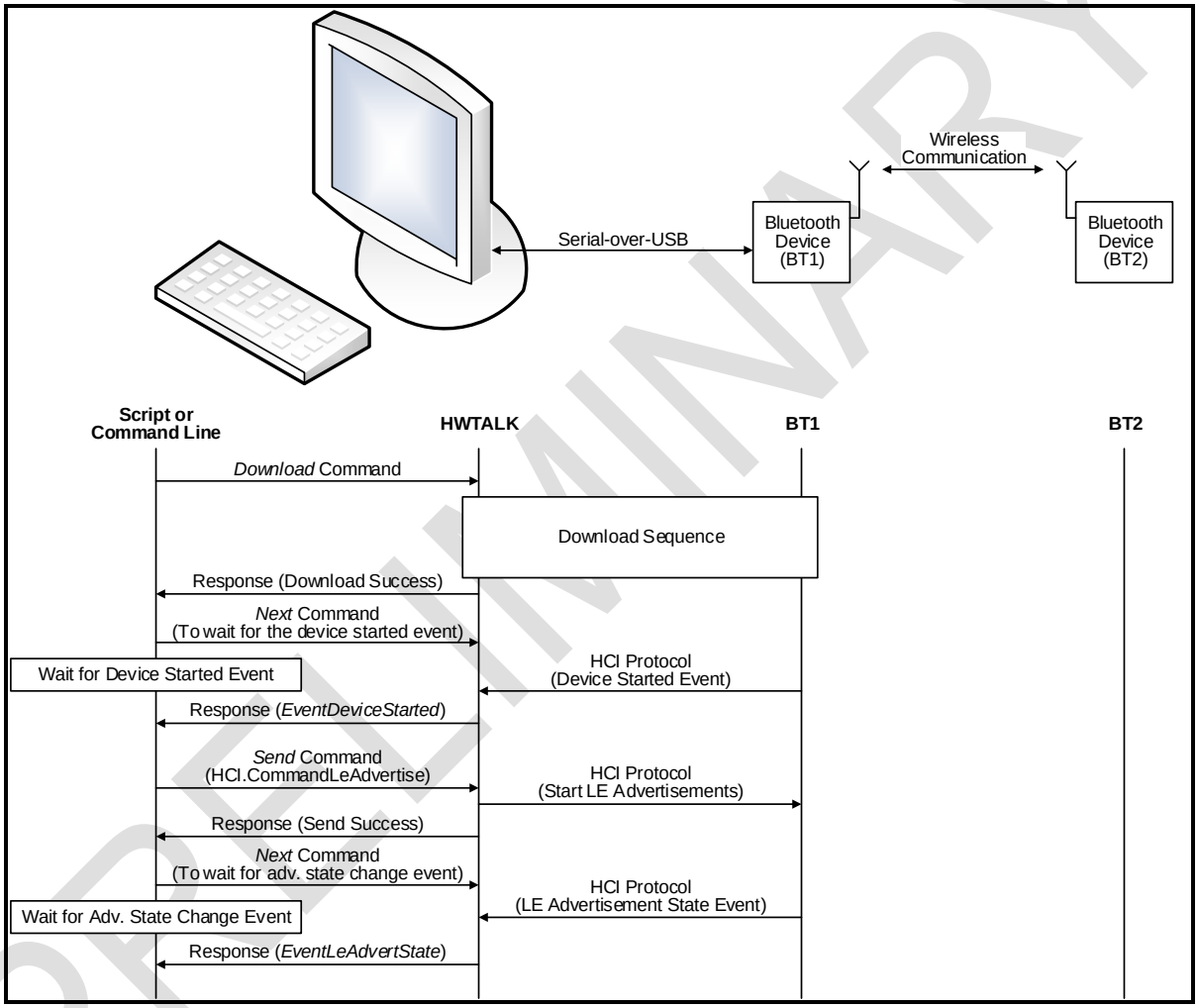
Exit (see “Exit”)

Download (see “Download”)

Sequences using these commands are typically scripted but can be entered manually as well. To understand how they are used within scripts, see Example Scripts.

Figure 4 shows a sample command/response message flow after HWTALK is started.

Figure 4. Sample Command/Response Message Flow



In Figure 4, HWTALK is used to download an application to a device (BT1), wait for the device to start, enable device advertisements after receiving EventDeviceStarted, and wait for the event indicating that advertisements have started (EventLeAdvertState). For more information on the Download Sequence shown in the figure, see [1].

6.2.1 Send

The Send command causes HWTALK to create and send a WICED HCI Command packet to the CYW20706 device over the configured serial connection.

The syntax of the Send command is:

```
send <structureName> [<field1>=<val1>, | <field2>=<val2>, | ... <fieldN>=<valN>]
```

HWTALK responds to the command with one of the following strings:

PRELIMINARY

- success
- error

In this send command syntax, `structureName` is the name of a command structure defined in the *wicedhcicmd.sdl* file prepended with the `HCI.` prefix, and the `<fieldN>=<valN>` pairs represent the relevant parameter names and parameter values of a specific command structure.

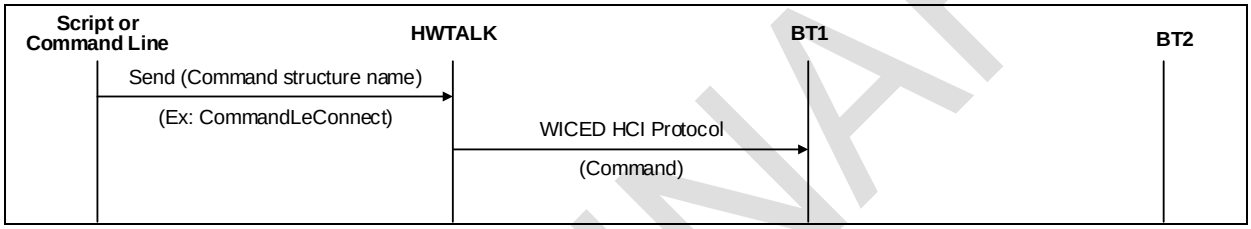
For example, if `structureName` is `CommandLeAdvertise`, then there is one parameter pair according to the `CommandLeAdvertise` structure in the *wicedhcicmd.sdl* file. The parameter is named "Cmd", and it can be assigned a value of '0' (to stop advertisements) or '1' (to start advertisements).

To start advertisements (for example), use the following Send command:

```
send HCI.CommandLeAdvertise Cmd=1
```

Figure 5 shows a command/response message flow for the Send command.

Figure 5. Message Flow for the Send Command



6.2.2 Next

When HWTALK receives a WICED HCI event from a CYW20706 device, it stores the packet in an internal buffer.

The Next command causes HWTALK to move the internal-buffer pointer to the next buffered event packet or wait for the specified timeout in milliseconds if the buffer is empty. When a pointer is set to a specific buffer, the script can extract different fields using the Get command (see "Get").

The syntax of the Next command is:

```
next <timeout>
```

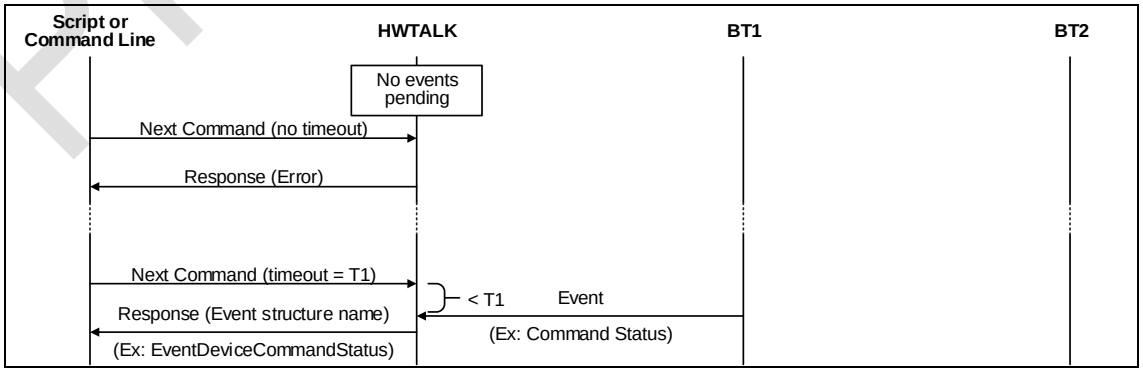
HWTALK responds to the command in one of the following ways:

- It returns the structure name of a received event. (See the *wicedhcievt.sdl* file to see all possible event structure names and definitions.)
- It returns the error string if the buffer is empty and no event is received within the specified timeout value.

To get data from a WICED HCI event after issuing a Next command, use the Get command (see "Get").

Figure 6 shows some typical command/response message flows for the Next command.

Figure 6. Typical Message Flows for the Next Command



PRELIMINARY

6.2.3 Get

The Get command gets the value of a field from the WICED HCI event structure currently pointed to by the internal-buffer pointer. Before issuing a Get command, issue a Next command (see "Next") to first move the internal-buffer pointer to the required buffered event packet.

The syntax of the Get command is:

```
get <fieldName>
```

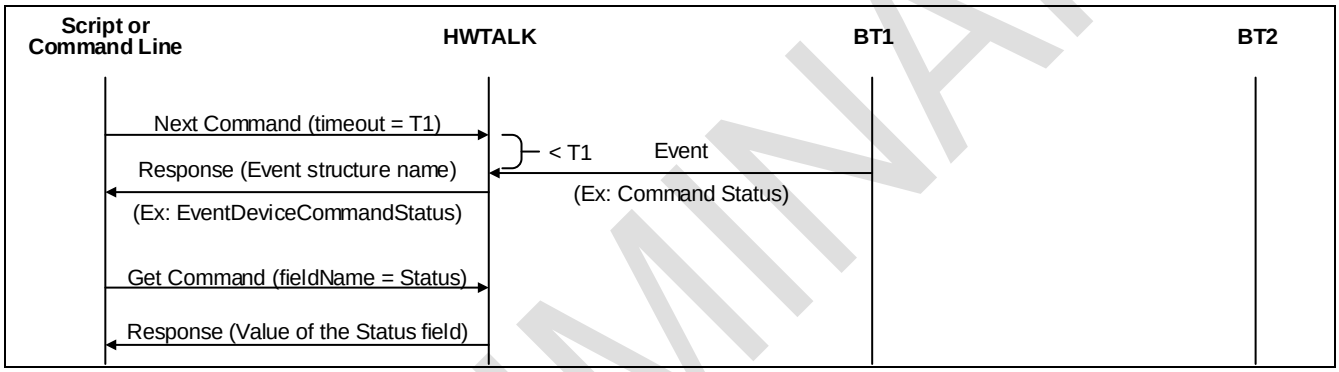
`fieldname` must match a parameter defined in the *wicedhcievt.sdl* file for the structure name returned after issuing the Next command.

HWTALK responds to the command in one of the following ways:

- It returns the value of the parameter whose name matches the fieldname.
- It returns the error string if no parameter matches the fieldname.

Figure 7 shows a typical command/response message flow for the Get command.

Figure 7. Typical Message Flow for the Get Command



6.2.4 Exit

The syntax of the Exit command is:

```
exit
```

The Exit command instructs HWTALK to complete the execution and release the serial connection.

6.2.5 Download

The syntax of the Download command is:

```
download <filePathName>
```

The Download command can be used to download firmware to test applications running on CYW20706 modules that do not have serial flash. If a CYW20706-based device has internal serial flash, then do not use the Download command.

On receiving the command, HWTALK does the following:

- Resets the device using the 'HCI_Reset' command
- Configures the serial connection baud rate for 3 Mbps
- Downloads the embedded application to be tested
- Issues the 'Launch RAM' command to start the application

If the download is successful, HWTALK sends the success string to STDOUT; otherwise, it sends the error string.

7 Structure Definition Language Files

The following two Structure Definition Language (SDL) files (also referred to as Message Definition files) are supplied with the WICED HCI Test system:

PRELIMINARY

- The *wicedhcicmd.sdl* file defines the WICED HCI command structures that apply to the Send command (see "Send").
- The *wicedhcievt.sdl* file defines the WICED HCI event structures that apply to Next and Get events (see "Next" and "Get").

The command and event structures in the SDL files are quite intuitive; they match the formats described in WICED Sample HCI UART Control Protocol (see [1]). For example, the LE Advertise command is defined as a command with an opcode of '2' and a 1-byte `Cmd` parameter indicating whether to start or stop advertisements.

In the SDL file, the command structure for starting or stopping advertisements is defined as follows:

```
/* start advertisements */
structure CommandLeAdvertise extends CommandLe
{
  option Command = [2]; option Length = [1];

  required uint8 Cmd; //* 0 - Stop Advertisements; 1 - Start Advertisements */
};
```

In this code excerpt, the `extends` keyword indicates that the `CommandLeAdvertise` structure is based on (or is an extension of) the structure of the LE Advertise command (which is indicated as `CommandLe` in the snippet). The LE Advertise command has an opcode of '2' and a parameter for starting or stopping advertisements. The extended `CommandLeAdvertise` structure includes a length value of '1' (indicating that the command has one byte).

The `CommandLe` structure is an extension of the `WicedPacket` structure with a `Group` code of '1'.

```
abstract structure CommandLe extends WicedPacket
{
  option Group = [1];
};
```

The `WicedPacket` structure is a 5-byte array with the first octet set to 25.

```
abstract structure WicedPacket
{
  required uint8 PacketType = [25]; required uint8 Command;
  required uint8 Group;
  required uint16 Length;
};
```

The `Command`, `Group`, and `Length` parameters are defined by the structures that extend `WicedPacket` as shown above. Based on this information, the SDL file defines the `CommandLeAdvertise` structure as the following array of bytes:

25	2	1	1	0	0 or 1

The highlighted two-byte length must be in little-endian format.

The following code shows how to add a new command to send a vendor-specific packet with a `Group` code of 255 and an opcode (that is, `Command` value) of 40 for passing three bytes of data:

```
abstract structure CommandVendorSpecific extends WicedPacket
{
  option Group = [255];
};
structure CommandVendorSpecificCommand40 extends CommandLe
{
  option Command = [40];
  option Length  = [3];
  required uint8 Param[3];
};
```

PRELIMINARY

8 Example Scripts

Scripts can be used to send HWTALK commands and receive events. WICED Studio comes with some example Perl and Python scripts.

8.1 Perl

WICED Studio comes with the *example test.pl* Perl test scriptm which does the following:

- Downloads the `hci_av_source_plus.hcd` application via UART to the attached CYW20706 device
- Waits for the Device Started event
- Sends a command to start an LE scan
- Waits for the LE Scan Status changed event

The *test.pl* code is shown below.

```
use IPC::Open2;

# sends first parameter to the HWTALK's stdin and returns response received from hwtalk's
# stdout if event is received from HWTALK, the procedure strips the white space characters
# (LFCR) at the end of received string and returns the buffer to the caller.
sub sendReceive {
    print Writer "$_[0]\n";
    chomp(my $str = <Reader>);
    # if hwtalk exited then return error
    if (!defined $str) {
        $res = "error";
    }
    # remove white space (LFCR) from the end of response
    $str =~ s/\s+$//;
    return $str;
}

# gets received messages till named in first parameter waiting if needed but no longer then
# the value of the second parameter with default 1 sec (1000)
# returns the name of the requested message or error
sub getWait {
    my $name = shift;
    my $timeout = shift;
    my $elapsed = 0;
    my $str;
    if(!defined $timeout) {
        %timeout = 1000;
    }
    while(1) {
        #get next message which is received already (no wait)
        $str = sendReceive("next");
        #if no received messages then get next message with 100ms wait
        if($str eq "error") {
            $str = sendReceive("next 100");
        }
        #exit if it is expected message
        if($str eq $name) {
            last;
        }
        #exit if timeout is elapsed
        $elapsed += 100;
        if($elapsed > $timeout) {
            $str = "error";
            last;
```

PRELIMINARY

```
        }
    }
    return $str;
}

# start and initialize hwtalk taking port from the command line
local (*Reader, *Writer);
$pid = open2(\*Reader, \*Writer, "hwtalk.exe -PORT " . $ARGV[0] . " -MSGDEFS wicedhcicmd.sdl -
MSGDEFSEVT wicedhcievt.sdl -LOGTO log.txt");

my $res;
# send command to execute "download" procedure passing "hci_av_source_plus.hcd" as a parameter
$res = sendReceive("download hci_av_source_plus.hcd");

# if something wrong happens HWTALK will return an "error" string
if($res ne "error") {
    #wait for end of device startup
    $res = getWait("EventDeviceStarted", 1000);
    if($res ne "error") {
        # start LE scan.  Send HCI.CommandLeScan with single parameter Cmd=1
        my $res = sendReceive("send HCI.CommandLeScan Cmd=1");
        if($res ne "error") {
            #wait for scan status event
            $res = getWait("EventLeScanStatus", 1000);
            if($res ne "error") {
                #get status field from the event and print it
                $res = sendReceive("get Status");
                if($res ne "error") {
                    print "Status = $res\n";
                }
            }
        }
    }
}

# tell hwtalk to exit
print Writer "exit\n";
close Writer;
close Reader;
waitpid($pid, 0);
```

8.2 Python

The WICED HCI test system comes packaged with some sample Python scripts. The path to the Python scripts is: <WICED-Studio>\test\xsdl\python_test_scripts.

To simplify Python scripting, import the `wiced_bt_class` Python class defined in *python_test_scripts\library\wiced_bt.py* and use the methods defined in the class.

9 References

[1] AN216618 - WICED HCI Control Protocol

[2] AN216535 - CYW92070xV3_EVAL Hardware User Manual

[3] AN218191 - WICED Quick Start Guide for BT CYW20706

[4] Bluetooth Core Specification, Version 4.2 (https://www.bluetooth.com/specifications/adopted-specifications)

PRELIMINARY

Document History

Document Title: AN216655 – WICED HCI Test System

Document Number: 002-16655

Revision	Submission Date	Description of Change
*A	01/12/2017	Updates to template and formatting - Preliminary
**	10/18/2016	Initial release - Preliminary

PRELIMINARY

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2016. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.

PRELIMINARY