Computer Access Technology Corporation

2403 Walsh Avenue, Santa Clara, CA 95051-1302 Tel: +1/408.727.6600 Fax: +1/408.727.6622

CATC Merlin's Wand™ 2.00

Bluetooth™ Test Generator

User's Manual

Document Revision 2.00

May 15, 2003

730-0019-00

CATC Merlin's Wand 2.00 Bluetooth Test Generator User's Manual, Document Revision 2.00

Document Disclaimer

Trademarks and Servicemarks

Copyright

Part number: 730-0019-00

Merlin's Wand Conformance Statements

FCC Conformance Statement

This equipment has been tested and found to comply with the limits for both a Class A and Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial or residential environment. This equipment generates, uses, and can radiate radio frequency energy, and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. The end user of this product should be aware that any changes or modifications made to this equipment without the approval of CATC could result in the product not meeting the Class A or Class B limits, in which case the FCC could void the user's authority to operate the equipment.

Important Notice
This equipment contains Transmitter Module FCC ID PNI8001001. To comply with FCC RF exposure requirements (sections 1.1307 and 1.310 of the Rules) only the antenna supplied by CATC must be used for this device. The antenna must be located at least 20 centimeters away from all persons.

EU Conformance Statement

This equipment complies with the R&TT Directive 1999/5/EC. It has been tested and found to comply with EN55022:1998 Class B (EN61000-3-2:1998, EN61000-3-3:1995), EN55024:1998 (EN61000-4-2:1995, EN61000-4-3:1995, EN61000-4-4:1995, EN61000-4-5:1995, EN61000-4-6:1995, EN61000-4-11:1994), and EN60950:1999. The transmitter module was tested and found to comply with ETS 300 328 (1997).

TABLE OF CONTENTS

1. Overview

The CATC Merlin's Wand™ Bluetooth™ Test Generator is a member of CATC's industry-leading line of Bluetooth protocol analysis tools and test equipment.

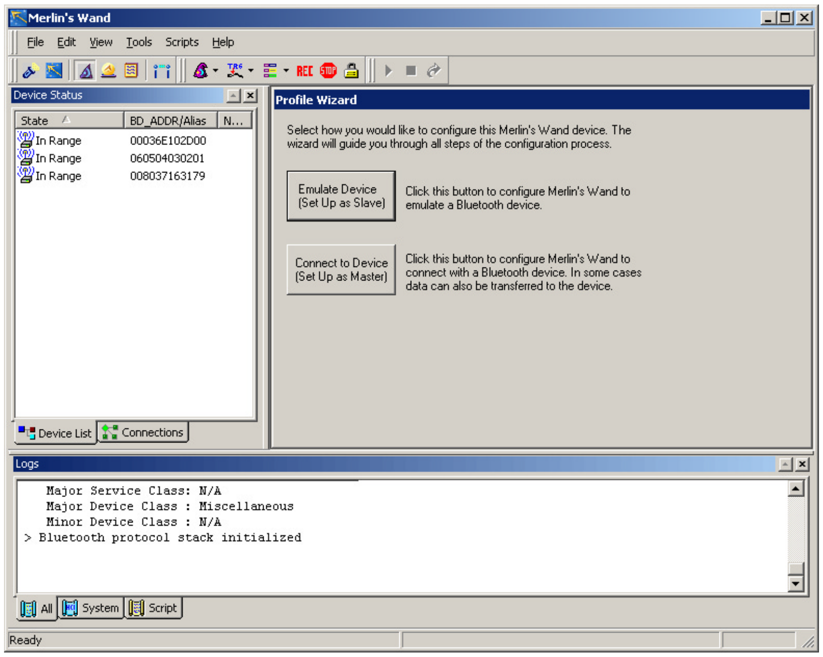
Preceded by CATC's Merlin™, a Bluetooth Protocol Analyzer, Merlin's Wand has been designed as an intelligent Bluetooth wireless technology device that can be used as a verification/validation tester or as an engineering debug and analysis tool. Through its software interface, designers and test technicians will be able to quickly and easily issue protocol commands and test sequences to analyze or validate designs to ensure compliance to the Bluetooth specification. Merlin's Wand can be used in conjunction with the Merlin protocol analyzer, allowing for real-time captures of test sequence results, as is required by the Bluetooth SIG to provide evidence of product compliance to the specification.

1.1 Applications

Merlin's Wand is a combination of hardware and Microsoft® Windows®-based application software. The hardware/software combination is capable of acting as a standard Bluetooth master or slave device within a piconet. By allowing this capability, Merlin's Wand can be used to establish or participate in a piconet and to send or receive data within the piconet. Through the Merlin's Wand Profile Wizard, users can quickly and easily manage Bluetooth wireless traffic generation. Additionally, via its Command Generator, Merlin's Wand can issue individual Bluetooth commands to a device under test, allowing a designer to focus his or her effort on a specific function or group of functions related to the device. Furthermore, users can quickly create test sequences with Script Manager, thus eliminating the difficulties normally associated with the creation of complex test sequences.

1.2 Merlin's Wand User Interface

The Merlin's Wand user interface consists of the Main window, the Logs window at the bottom of the screen, and the Device Status window on the left side of the screen.



State	BD_ADDR/Alias	N...
In Range	00036E102D00	
In Range	060504030201	
In Range	008037163179	

Select how you would like to configure this Merlin's Wand device. The wizard will guide you through all steps of the configuration process.

Click this button to configure Merlin's Wand to emulate a Bluetooth device.

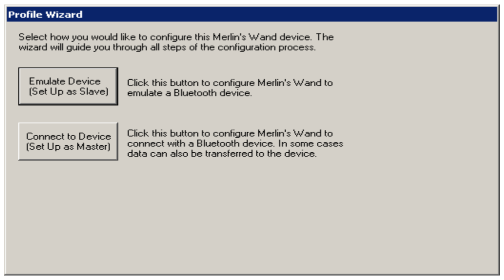
Click this button to configure Merlin's Wand to connect with a Bluetooth device. In some cases data can also be transferred to the device.

```
    Major Service Class: N/A
    Major Device Class : Miscellaneous
    Minor Device Class : N/A
> Bluetooth protocol stack initialized
```

The application's primary tools are run within the Main window: *Profile Wizard*, *Command Generator*, and *Script Manager*.

Each tool offers a different means of generating traffic.

Note: Only one tool can be run at a time.


Profile Wizard
Emulate Device (Set Up as Slave)
Connect to Device (Set Up as Master)

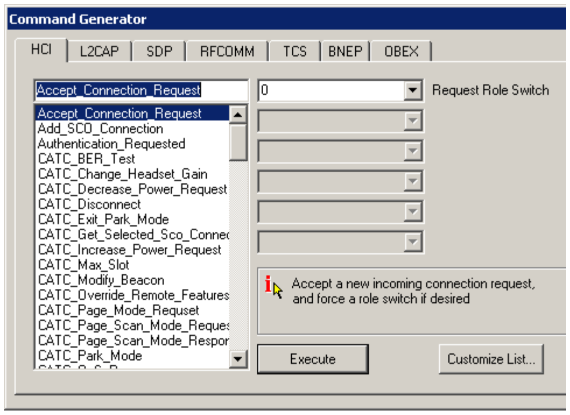
Select how you would like to configure this Merlin's Wand device. The wizard will guide you through all steps of the configuration process.

Click this button to configure Merlin's Wand to emulate a Bluetooth device.

Click this button to configure Merlin's Wand to connect with a Bluetooth device. In some cases data can also be transferred to the device.

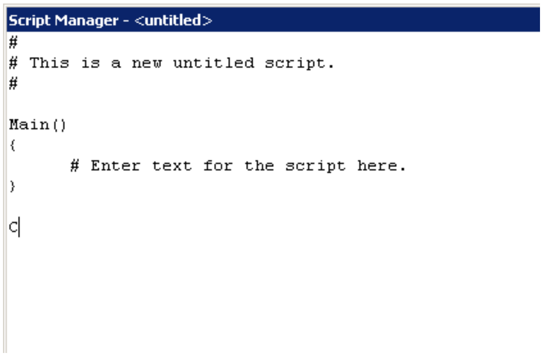
Profile Wizard is a point-and-click tool for creating connections and transferring data between Merlin's Wand and other Bluetooth devices. This tool requires little Bluetooth wireless technology experience and allows you to generate Bluetooth traffic without having to execute specific Bluetooth commands. Profile Wizard is described in Chapter 3, *Profile Wizard*, on page 17.

Command Generator is a tool that presents a menu of protocol commands that can be selected and executed in virtually any sequence. Command Generator thus offers maximum control over the traffic generation process, but also requires familiarity with the Bluetooth commands. Command Generator is described in Chapter 4, *Command Generator*, on page 43.



Accept a new incoming connection request, and force a role switch if desired

Script Manager is a tool that provides an editor for writing and/or executing scripts that will generate Bluetooth wireless traffic. With Script Manager, new scripts can be written and saved, or existing scripts may be opened, edited, and run. Script Manager is described in Chapter 5, *Script Manager*, on page 55.



```
#
# This is a new untitled script.
#

Main()
{
        # Enter text for the script here.
}

c
```

1.3 Key Features

- Plug-and-play USB connection between test system and test module
- External antenna can be removed to create wired piconet
- Audio connector for connecting audio devices, such as headsets
- Can operate as either a master or slave device in a piconet
- Graphical interface allows for easy selection of command parameters
- Wizard provided to reduce learning curve and memorization of command sequences
- Test modes provide for these Bluetooth wireless technology protocols: HCI, L2CAP, SDP, RFCOMM, TCS, BNEP, and OBEX
- Scripting capability for establishment of predefined test sequences
- System information report provides details regarding device under test

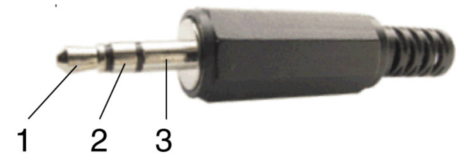
- Can be used with Merlin protocol analyzer
- Power-on self-diagnostics
- No external power required -- obtains power from USB connection
- One year warranty and online customer support

Please refer to the *Bluetooth Specification, version 1.1* for details on the Bluetooth wireless technology protocol. The Bluetooth specification is available from the Bluetooth SIG at its web site **http://www.Bluetooth.com**.

1.4 Audio Connections - Important Information

Merlin's Wand has a 2.5 mm audio stereo jack for plugging in headsets.

Headsets need to have a 2.5 mm plug with the following pinout:



1) **Microphone** (signal from headset; bias power of 2.5 V and maximum 1 mA provided by Merlin's Wand on the same pin)

2) **Speaker** (signal to headset; speaker impedance needs to be >16 Ohm)

3) **Ground**

The following headsets have been successfully tested with Merlin's Wand:

- **RadioShack**® 43-1957 Super Lightweight Hands-Free Headset
- **GN Netcom**® GNX Mobile M200
- **Motorola**® Retractable Hands-Free Headset Model # 98196G
- **Belkin**® Universal 2.5 mm Personal Hands-Free Kit F8V920-PL
- **Coby**® CV-M20 Earphone with Built-In Microphone
- **Plantronics**® CHS122N Hands-Free Headset
- **Plantronics** M110 Headset for Cordless and Mobile Phones

1.5 Limitations

- The only Inquiry Access Code (IAC) supported at inquiry and inquiry scan is the General Inquiry Access Code (GIAC)
- The Scan_Enable parameter value of 0x01 (inquiry scan enabled and page disabled) is not supported
- Page scan intervals and inquiry scan intervals other than 1.28s are not supported
- Page scan windows and inquiry scan windows other than 11.25ms are not supported
- Optional page scan modes are not supported
- More than one SCO connection at a time is not supported

1.6 Specifications

The following specifications describe a Merlin's Wand System.

Package

Dimensions:	3.4 x 2.6 x 1 inches (8.6 x 6.6 x 2.54 centimeters)
Connectors:	Host connection (USB, type 'B') Audio connection (2.5 millimeter audio stereo jack)
Weight:	3.0 oz. (84 g)

Environmental Conditions

Operating Range:	0 to 55 °C (32 to 131 °F)
Storage Range:	-20 to 80 °C (-4 to 176 °F)
Humidity:	10 to 90%, non-condensing

Host Compatibility

Works with any PC equipped with a functioning USB port and a Microsoft Windows 98 SE, Windows Me, Windows 2000, or Windows XP operating system.

Hardware Interfaces

Standard USB Interface -- connects to the host computer
2.4 GHz (ISM band) External Antenna.
2.5 mm audio stereo jack

Product Warranty

CATC offers a one-year limited warranty on its products.

2. Getting Started

This chapter describes how to install Merlin's Wand and its software. Both install easily in just a few minutes. The Merlin's Wand software can be installed on most Windows-based personal computer systems.

2.1 System Requirements

The following is the recommended configuration for the computer that runs the Merlin's Wand application and is connected to the Merlin's Wand hardware unit.

- **Operating system**: Microsoft® Windows® 98 SE, Windows 2000, Windows Me, or Windows XP operating system.

  Occasionally, after unplugging the Merlin's Wand hardware on a Windows 2000 system, subsequent attempts to plug in the device cause the computer to recognize the unit as a "USB device" rather than as "Merlin's Wand." If this occurs, you will need to restart the computer so that it will recognize the device properly. To avoid the problem, upgrade to the latest Windows 2000 service pack available from Microsoft.

- **Required setup**: Microsoft Internet Explorer 5 or later must be installed.
- See **readme** document on installation CD or in installation directory for latest hardware requirements.

2.2 Setting Up Merlin's Wand

The Merlin's Wand hardware can be set up using the installation CD-ROM or from installation files downloaded from the CATC website.

Step 1 Attach the external antenna to the Merlin's Wand hardware unit by screwing it onto the connector labelled ANT.

Step 2 Plug one end of the USB cable into the USB port on the Merlin's Wand hardware, and plug the other end of the USB cable into a USB port on the host computer.

Windows should automatically detect the Merlin's Wand hardware and open the Windows Hardware Wizard to install Merlin's Wand. If the installation doesn't finish automatically, proceed to Step 3.

Step 3 Follow the Hardware Wizard's on-screen instructions to complete the installation. If the wizard prompts you for driver information, insert the Merlin's Wand installation CD-ROM and direct the wizard to the directory *<drive>*:\software\MerlinWand122. Substitute the drive letter of the CD-ROM drive for <*drive*>. For example, if the CD-ROM drive is drive D, navigate to D:\software\MerlinWand122.

Note: If you are using installation files downloaded from the CATC website, you will need to direct the wizard to the Disk 1 directory of the installation files so that it can locate the driver.

2.3 Installing the Software and Starting the Program

The Merlin's Wand software can be installed from the installation CD-ROM or from installation files downloaded from the CATC website.

Install from CD-ROM

Step 1 Insert the Merlin's Wand installation CD-ROM into the CD-ROM drive of the computer that will be connected to the Merlin's Wand hardware.

The autorun program should start automatically. If it doesn't start, use Windows Explorer or My Computer to navigate to the CD-ROM drive directory and double-click the file **autorun.exe,** and proceed to Step 2. If it still doesn't start, navigate to the \software\MerlinWand122 directory on the CD-ROM, double-click the file Setup.exe, and proceed to Step 3.

Step 2 Choose **Install Software** to start the setup program.

Step 3 Follow the on-screen instructions to complete the installation.

Install from installation download

Step 1 Select **Start > Run...** from the Windows taskbar and click the **Browse** button, then navigate to the Disk 1 directory of the Merlin's Wand installation download. Select the file **Setup.exe** and click **Open**.

Step 2 Follow the on-screen instructions to complete the installation.

Start the program

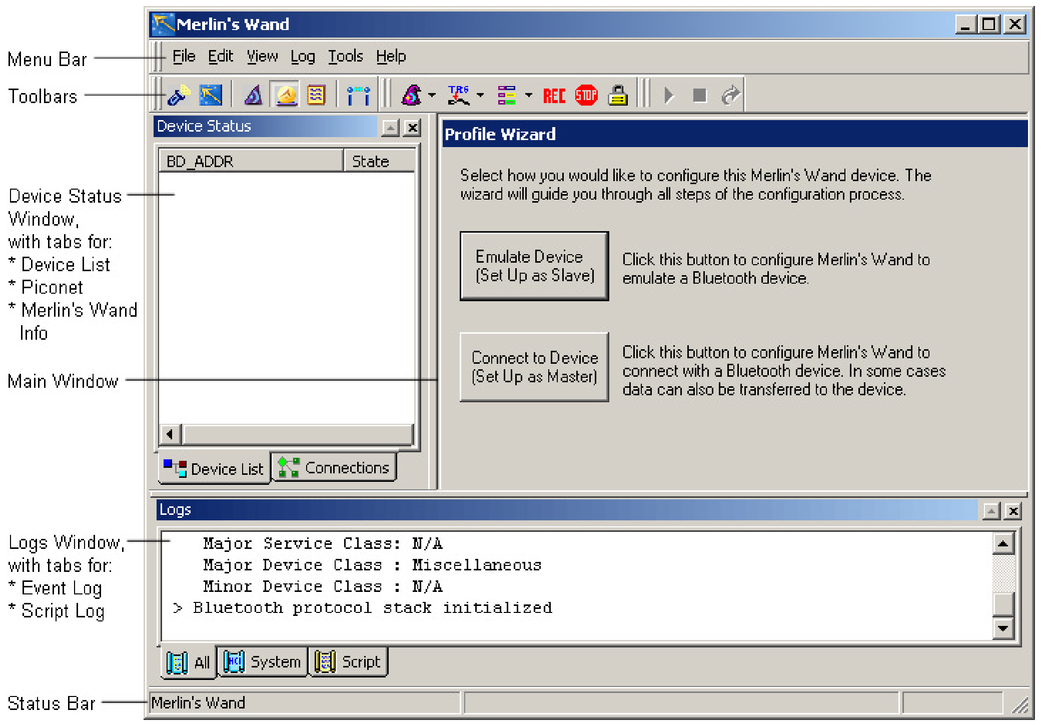
Once the software has been installed, be sure that the Merlin's Wand hardware is connected to the PC via the USB cable before starting the Merlin's Wand application. Otherwise, the application will provide a warning message telling you that the Wand could not be found.

To start the application, select **Start > Programs > CATC > CATC Merlin's Wand**. Note that this is the default location for the Merlin's Wand application. If it was installed in a different folder, select that folder from the Programs menu.

2.4 Displaying the On-Screen Help

Access the on-screen Help included with the Merlin's Wand application by selecting **Help > Help...** from the menu bar.

2.5 Application Layout


Menu Bar
Toolbars
Device Status Window, with tabs for: * Device List * Piconet * Merlin's Wand Info
Main Window
Logs Window, with tabs for: * Event Log * Script Log
Status Bar
Merlin's Wand
File Edit View Log Tools Help
Device Status
BD_ADDR
State
Device List
Connections
Profile Wizard
Emulate Device (Set Up as Slave)
Connect to Device (Set Up as Master)
Logs
All
System
Script
Merlin's Wand

Select how you would like to configure this Merlin's Wand device. The wizard will guide you through all steps of the configuration process.

Click this button to configure Merlin's Wand to emulate a Bluetooth device.

Click this button to configure Merlin's Wand to connect with a Bluetooth device. In some cases data can also be transferred to the device.

```
    Major Service Class: N/A
    Major Device Class : Miscellaneous
    Minor Device Class : N/A
> Bluetooth protocol stack initialized
```

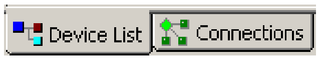
The Merlin's Wand window is made up of the following:

- The Main window, where the primary tools are run: **Profile Wizard**, **Command Generator**, and **Script Manager**.
  - ***Profile Wizard*** -- A simple, easy-to-use tool that guides you through the process of establishing connections and generating traffic between Merlin's Wand and other Bluetooth wireless technology devices.

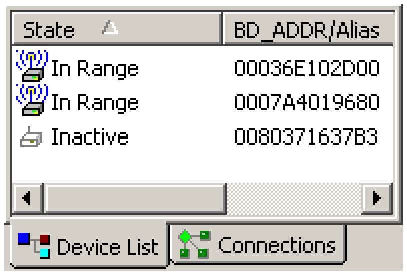
- ***Command Generator*** -- A tool that allows Bluetooth commands to be issued in any chosen sequence. If Command Generator isn't enabled on your Merlin's Wand system, a License Key must be obtained from CATC before it can be used.
- ***Script Manager*** -- A notepad-like tool for writing and launching scripts that cause Merlin's Wand to generate traffic. This tool is an optional feature. If Script Manager isn't enabled on your Merlin's Wand system, a License Key must be obtained from CATC before it can be used.

Note: When switching between Profile Wizard, Command Generator and Script Manager, all connections that have been established between Merlin's Wand and another Bluetooth device should be closed. However, expert users may choose to leave the connections open. If a connection is left open and you attempt to switch tools, Merlin's Wand will prompt you to close the connections. Choosing **Disconnect All** will close the connections. Choosing **Cancel** will leave the connections open, but some commands might not work properly in the other tool. When switching to Profile Wizard, any open connections *must* be closed.

- The **Device Status** window is on the left side of the interface. It contains two tabs: **Device List** and **Connections**.



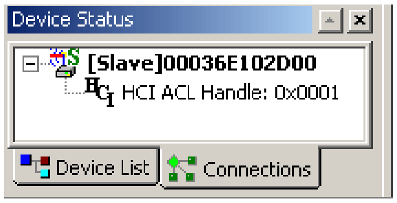
  - ***Device List*** - Displays a list of devices that Merlin's Wand has discovered. It also contains information about the devices found, such as the Bluetooth address, the state, the role, the class, and the device's local name.



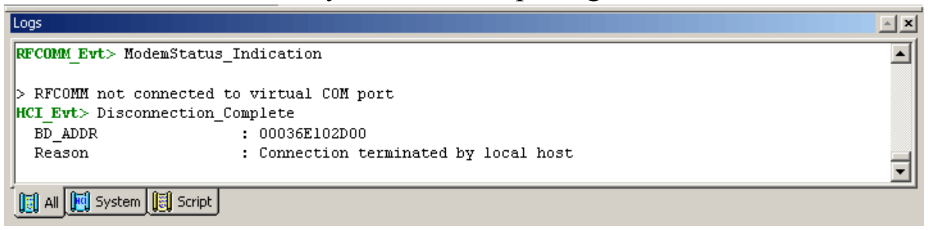
State	BD_ADDR/Alias
In Range	00036E102D00
In Range	0007A4019680
Inactive	0080371637B3

This window is open by default. These symbols in the list indicate a device's state: **C** = Connected; **i** = In Range. Right-clicking on a listed device opens the **Device List Pop-Up Menu**. The menu presents the following options: *Connect*, *Add Audio Connection*, *Get Device Information*, *Delete*, and *Disconnect All*. For details on using the Pop-Up Menu, see Chapter 6, *Device Search and Device List Pop-Up Menu*, on page 61.

- ***Connections*** - Displays a hierarchical list of all connections between Merlin's Wand and other devices. At the top of the list is the address of the connected device; below it are the various channels established between Merlin's Wand and the device. Symbols: **C** = Connection; **H** = HCI ACL; **S** = HCI SCO; **L** = L2CAP; **R** = RFCOMM; **O** = OBEX.

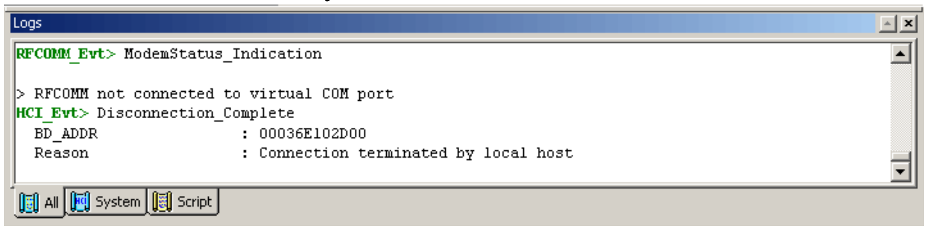


- At the bottom of the interface is the **Logs window**, which contains tabs for the Event Log and the Script Log:
  - ***All Log*** - Captures the commands and events of the System and Scripts logs



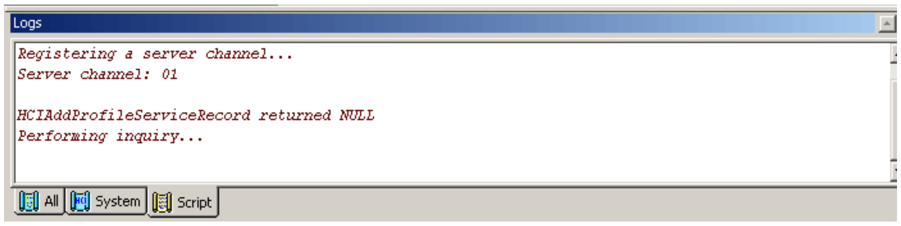
```
RFCOMM_Evt> ModemStatus_Indication

> RFCOMM not connected to virtual COM port
HCI_Evt> Disconnection_Complete
  BD_ADDR                 : 00036E102D00
  Reason                  : Connection terminated by local host
```

  - ***System Log*** - Maintains a log of all commands issued by Merlin's Wand and the events that ensue, such as a reply by another device.



```
RFCOMM_Evt> ModemStatus_Indication

> RFCOMM not connected to virtual COM port
HCI_Evt> Disconnection_Complete
  BD_ADDR                 : 00036E102D00
  Reason                  : Connection terminated by local host
```

  - ***Script Log*** - Maintains a record of the commands issued by Script Manager and the events resulting from these commands. If line numbers are referenced in the Script Log, double-clicking on the line number will move the cursor to that line in the Script Manager window.



```
Registering a server channel...
Server channel: 01

HCIAddProfileServiceRecord returned NULL
Performing inquiry...
```

2.6 Menus

The menu bar at the top of the application window contains the following menus of pull-down commands:

Table 1: Menu Bar Commands

File Menu

Command	***Function***
New	Creates a new script file
Open...	Opens a script file
Close	Closes a script file
Save	Saves a script file
Save As..	Saves a script file with a specified name
Print Setup...	Sets up the current or a new printer
Print Script...	Prints a script file
Exit	Exits the Merlin's Wand program

Edit Menu

Command	***Function***
Undo	Undoes last change
Cut	Cuts text
Copy	Copies text
Paste	Pastes copied or cut text
Select All	Selects all text
Find...	Finds specified string
Find Next	Repeats last find action
Replace...	Searches for a string and replaces it with a new string

View Menu

Command	***Function***
Device Status	Shows or hides the Device Status window
Logs	Shows or hides the Logs window
Toolbars >	Opens sub-menu with options for showing or hiding the Standard, Tools and Modes, Merlin/Merlin Mobile, and Scripts toolbars
Status Bar	Shows or hides the status bar
Intro	Displays Introduction splash screen that displays when the application first opens

Tools Menu

Command	***Function***
Device Search	Opens the Device Search dialog box

Table 1: Menu Bar Commands (Continued)

Profile Wizard	Opens Profile Wizard
Command Generator	Opens Command Generator
Script Manager	Opens Script Manager
Data Transfer Manager	Opens Data Transfer Manager
Local Device Manager	Opens a dialog box that displays information about Merlin's Wand

Script Menu

Command	***Function***
Run	Runs the open script. Requires that Script Manager be running.
Stop	Stops execution of the open script. Requires that Script Manager be running.

Help Menu

Command	***Function***
Help...	Displays online Help
About Merlin's Wand	Displays version information about Merlin's Wand.

2.7 Toolbars

There are four toolbars in the Merlin's Wand user interface: Standard, Tools and Modes, Merlin/Merlin Mobile, and Scripts. The Toolbar buttons provide access to frequently-used program functions. Tool tips describe icon functionality as the mouse pointer is moved over an item.

Standard

These buttons require Script Manager to be open.

New Script

Opens a new page in Script Manager

Opens Script

Opens a dialog box for selecting a script

Save Script

Opens a Save As dialog box

Print Script

Prints open script

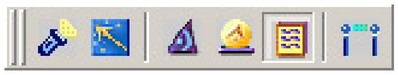
Cut

Cuts selected text in Script Manager

Insert Clipboard Contents

Pastes text from clipboard

Tools and Modes



Device Search

Opens Device Search dialog

Local Device Manager

Displays info about Merlin's Wand

Profile Wizard

Opens Profile Wizard

Command Generator

Opens Command Generator

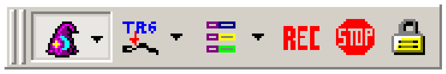
Script Manager

Opens Script Manager

Data Transfer Manager

Opens Data Transfer Manager dialog

Merlin/Merlin Mobile Analyzer Toolbar



Connect/Disconnect Merlin Bluetooth Analyzer

Connects to or disconnects from Merlin Bluetooth Analyzer

Set Merlin Recording Options

Displays the Open dialog to choose the Recording Options file for Merlin Bluetooth Analyzer

Set Merlin Display Options

Displays the Open dialog to choose the Display Options file for Merlin Bluetooth Analyzer

Start Recording

Starts a Merlin Bluetooth Analyzer recording session

Stop Recording

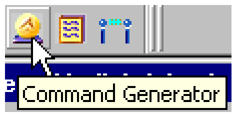
Stops a Merlin Bluetooth Analyzer recording session

Set Merlin Encryption Options

Opens the Encryption Setup window

2.8 Tool Tips

For most of the buttons and menus, tool tips provide useful information.



To display a tool tip, position the mouse pointer over an item. If a tooltip exists for the item, it will pop up in a moment.

2.9 Merlin's Wand Keyboard Shortcuts

Several frequently-used operations are bound to keyboard shortcuts.

Table 2: Keyboard Shortcuts

Key Combination	Operation	Key Combination	Operation
Ctrl + A	Select all	Ctrl + V	Paste
Ctrl + C	Copy	Ctrl + W	Close script
Ctrl + F	Find	Ctrl + X	Cut
Ctrl + G	Go to	Ctrl + Z	Undo
Ctrl + H	Replace	Home	Jump to first character of line
Ctrl + I	Indent	End	Jump to last character of line
Ctrl + N	New script	Ctrl + Home	Jump to first character of file
Ctrl + O	Open script	Ctrl + End	Jump to last character of file
Ctrl + P	Print script...	Ctrl + Backspace	Delete previous word
Ctrl + R	Run script	F3	Find next
Ctrl + S	Save script	Alt + F4	Shut down Merlin's Wand application

3. Profile Wizard

Once Merlin's Wand is installed and running, it is ready to generate traffic.

The easiest way to generate traffic is to use Profile Wizard, a point-and-click tool for creating connections and transferring data between Merlin's Wand and other Bluetooth wireless technology devices. This tool requires little Bluetooth experience and allows you to generate Bluetooth traffic without having to execute specific Bluetooth commands. Profile Wizard manages the entire traffic generation process. Just follow the on-screen instructions and Merlin's Wand will execute the Bluetooth commands needed to make the connection.

Profile Wizard allows you to configure Merlin's Wand to connect to or emulate several Bluetooth devices. Profile Wizard provides options for devices that use these profiles:

- Dial-Up Gateway
- Fax Gateway
- File Transfer
- Hardcopy Cable Replacement Profile (HCRP) Server
- Headset, Headset Audio Gateway
- Local Area Network Access (LAN)
- Object Push
- Personal Area Networking–Group Ad-Hoc Network (PAN–GN)
- Personal Area Networking–Network Access Point Profile (PAN–NAP)
- Serial Port

3.1 Starting Profile Wizard

Click the Profile Wizard icon or select Tools > Profile Wizard from the menu bar.

Profile Wizard's opening screen will be displayed in the main window.

3.2 Connecting to Devices

Choosing the "Connect to Device" option on the opening screen of Profile Wizard allows you to configure Merlin's Wand to seek out other Bluetooth devices, connect to one of them, and possibly exchange data with it. This option causes Merlin's Wand to act as the master device.

NoteTo connect to a device that uses the Dial-Up Gateway, Fax Gateway, LAN, or Serial Port profile, you will need to install the virtual COM port driver that is included with the Merlin's Wand installation. For instructions, see "Installing the Virtual COM Port Driver" on page 39.

Step 1 Turn on the target device.

Step 2 Open Profile Wizard and click the **Connect to Device** button.

Merlin's Wand will perform a General Inquiry to collect information on local devices, then list the devices that are found on the Select Device screen.

(Optional) Uncheck "Skip Name Search" so that Merlin's Wand will not try to discover the devices' user-friendly names. This will speed up the search.

NoteIf no devices are found, you can click **Search** to have Merlin's Wand repeat the General Inquiry.

Step 3 *Select device from list*
From the list, select the device address to which you want Merlin's Wand to connect.

Add address
Click the **Add Address** button in order to use an address that does not appear in the list.

The Add Address dialog appears.

Enter the device address in 12-digit hexadecimal format. To enter a pin code to use with the device, check the option "For connections with this device, use the following PIN code" and then enter the code.

Click **OK** to close the dialog.

Add PIN code
To use a PIN code with a device that you selected from the list, click the **Enter PIN** button to open the Enter PIN dialog.

Enter the PIN for the device, then click **OK** to close the dialog.

Step 4 Click **Next**.

You will advance to the Configure Security screen.

Step 5 Select the security option that you want to apply for the device.

- *Do not apply authentication and encryption*.
  Authentication and encryption will not be used.

- *Apply authentication and encryption.*
  The default authentication and encryption settings will be used.
- *Use current authentication/encryption settings.*
  The authentication and encryption settings that are currently set in the Local Device manager will be used.
- *Display the Local Device Manager so I can configure security and verify PIN codes myself.*
  This option allows you to access the Local Device Manager to manually enter the security settings. If you choose this option, press Next to open the Local Device Manager. When you close the dialog, 6 will be performed automatically.

Step 6 Press **Next**.

Merlin's Wand will query the selected device to determine its profile. When the query is complete, the Select Profile screen opens and displays a list of profiles found.

You can now refer to the section that describes the setup for the particular profile that the target device uses:

- Section , "Connect to Device: Dial-Up Gateway" on page 19
- Section , "Connect to Device: Fax Gateway" on page 21
- Section , "Connect to Device: File Transfer" on page 22
- Section , "Connect to Device: HCRP Server" on page 23
- Section , "Connect to Device: Headset" on page 24
- Section , "Connect to Device: Headset Audio Gateway" on page 25
- Section , "Connect to Device: LAN" on page 26
- Section , "Connect to Device: Object Push" on page 26
- Section , "Connect to Device: PAN–GN" on page 27
- Section , "Connect to Device: PAN–NAP" on page 28
- Section , "Connect to Device: Serial Port" on page 29

Connect to Device: Dial-Up Gateway

Follow these steps to configure Merlin's Wand to connect to a Bluetooth device that uses the Dial-Up Gateway profile.

NoteTo connect to a device that uses the Dial-Up Gateway profile, you must install the virtual COM port driver that is included with the Merlin's Wand installation. For instructions, see "Installing the Virtual COM Port Driver" on page 39.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select Dial-up Gateway from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Click **Next**.

Merlin's Wand will establish a connection to the specified device.

Step 4 You can now send individual commands to the Dial-Up Gateway device via the Profile Wizard interface or send data by using an external communication application.

To send individual commands:

Use the Command combo box to manually enter a command or select one from the drop-down list. You can also select a command from the Command List and click **Add** in order to append the command to the current selection in the combo box.

Press **Send Cmd**. The selected command will be sent to the remote device.

Each new command or combination of commands that you send during the current session will be added to the drop-down list in the Command combo box.

To send data:

NoteA modem driver must be installed on the virtual COM port in order to use an external application to send data through Merlin's Wand. See "Install a Modem Driver on the Virtual COM Port" on page 40 to find out how to install the driver. Once the driver is installed, you must configure the external application to use that driver.

NoteAn external dial-up communications application (such as Windows Dial-Up Networking) must be installed on the host computer in order to perform this operation.

Open the application and send the data. It will use the modem driver on the virtual COM port to send the data to the Dial-Up Gateway device through Merlin's Wand.

Enable or disable event logging:
You can configure Merlin's Wand to write incoming or outgoing data to the Event Log by checking or unchecking the "Log Incoming Data" and "Log Outgoing Data" options.

Connect to Device: Fax Gateway

The Fax Gateway option allows you to use a fax application to send to data to Merlin's Wand via a virtual COM port. Merlin's Wand then uses Bluetooth to send the data to a device that uses the Fax Gateway profile to send data over a fax modem.

Before attempting this, you must install the virtual COM port driver that is included with the Merlin's Wand installation. For instructions, see "Installing the Virtual COM Port Driver" on page 39.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select Fax Gateway from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Click **Next**.

Merlin's Wand will establish a connection to the specified device.

Step 4 You can now send individual commands to the Fax Gateway device or use an external fax application to send data over the virtual COM port.

To send individual commands:
Enter a command in the combo box and press the **Send Cmd** button. Each command that you enter during the current session will be added to the drop-down list in the combo box.

To send fax data:

NoteA modem driver must be installed on the virtual COM port in order to use a fax application to send data through Merlin's Wand. See"Install a Modem Driver on the Virtual COM Port" on page 40 to find out how to install the driver. Once the driver is installed, you must create a fax printer and configure it to use the driver that you have installed.

NoteA fax application must be installed on the host computer in order to perform this operation.

Open the fax application and send a fax. It will use the modem driver on the virtual COM port to send the data to the Fax Gateway device through Merlin's Wand.

Enable or disable event logging:
You can configure Merlin's Wand to write incoming or outgoing data to the Event Log by checking or unchecking the "Log Incoming Data" and "Log Outgoing Data" options.

Connect to Device: File Transfer

These steps show how to configure Merlin's Wand to connect to and transfer data with a Bluetooth device that uses the File Transfer profile. This allows you to browse the directories on the device, transfer files to or from the device, and delete files or create new folders in the device's file system.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select File Transfer from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Click **Next**.

Merlin's Wand will establish an OBEX connection with the device, and the Wizard will advance to the File Transfer screen.

Step 4 You can now perform the following on either the remote device or the local device (Merlin's Wand):

- Create a new folder — To create a new folder in either device's file directory, right-click on the directory level in which you wish to place the new folder, then choose "Create new folder" from the menu that pops up.

  The Enter Folder Name dialog will appear. Type in a name for the new folder, then click **OK**. The new folder will appear in the chosen directory.

- Get or Put a file — To create a new folder in either device's file directory, right-click on the directory level in which you wish to place the new folder, then choose "Create new folder" from the menu that pops up.

  The Enter Folder Name dialog will appear. Type in a name for the new folder, then click **OK**. The new folder will appear in the chosen directory.

- Delete a file — To create a new folder in either device's file directory, right-click on the directory level in which you wish to place the new folder, then choose "Create new folder" from the menu that pops up.

  The Enter Folder Name dialog will appear. Type in a name for the new folder, then click **OK**. The new folder will appear in the chosen directory.

Connect to Device: HCRP Server

This section explains how to use Profile Wizard to configure Merlin's Wand to connect to a Hardcopy Cable Replacement Profile server, send commands to it, and send a file to print.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select HCRP Server from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Press **Next**.

Merlin's Wand will establish an L2CAP connection with the device, and the Wizard will advance to the HCRP client screen.

Step 4 You can send individual commands to the HCRP server, or specify a file to transfer to the server for printing.

Send an individual command:

(a) Select a command from the drop-down list.

(b) Click **Send Cmd**.

Select a file to transfer:

(a) Click **Browse**.
The Open dialog will come up.

(b) Use the Open dialog to browse the file that you want to transfer, then click **Open**.
The file's name and directory path will be shown in the Select File to Transfer for Printing box.

(c) Click **Start Transfer** to send the selected file to the server.

Connect to Device: Headset

The following steps show how to configure Merlin's Wand to connect to and transfer audio with a Bluetooth headset that uses the Headset profile.

Note In order to verify that Merlin's Wand and a Bluetooth audio device are successfully connected, a headset will need to be plugged into the audio port on Merlin's Wand. Be sure that the headset is plugged in *before* starting configuration and initializing the connection between Merlin's Wand and the headset.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select Headset from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen, and Merlin's Wand will attempt to establish an RFCOMM connection to the device.

If the connection attempt is successful, the Connection Status screen will show that Merlin's Wand has established an RFCOMM connection with the device. Merlin's Wand will automatically ring the target device and wait for an answer. Pressing the **Ring** button will cause Merlin's Wand to ring the device again.

When the target device answers, Merlin's Wand will establish an SCO connection with it.

Note If you cannot establish a connection, you can re-attempt the connection by either pressing Back and re-running the previous two steps, or by pressing the **Connect** button again.

Note The Speaker and Microphone Volume levels can be adjusted by moving the sliders up or down. The level is indicated by a number, from 0 to 15, to the left of each slider.

Step 4 (Optional) Click the **Disconnect** button on the Connection Status screen to close the connection.

The connection between Merlin's Wand and the device will terminate, and the **Connect** button will again be available. Selecting Connect will reestablish the connection.

Connect to Device: Headset Audio Gateway

The following steps show how to configure Merlin's Wand to connect to and transfer audio with a Bluetooth headset that uses the Headset Audio Gateway profile.

Note In order to verify that Merlin's Wand and a Bluetooth audio device are successfully connected, a headset will need to be plugged into the audio port on Merlin's Wand. Be sure that the headset is plugged in *before* starting configuration and initializing the connection between Merlin's Wand and the headset.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select Headset Audio Gateway from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen, and Merlin's Wand will attempt to establish a connection to the device.

If the connection attempt is successful, the Connection Status screen will show that Merlin's Wand has established a connection with the device.

Note If you cannot establish a connection, you can re-attempt the connection by either pressing Back and re-running the previous two steps, or by pressing the **Connect** button again.

Note The Speaker and Microphone Volume levels can be adjusted by moving the sliders up or down. The level is indicated by a number, from 0 to 15, to the left of each slider.

Step 4 To verify that Merlin's Wand and the Bluetooth device are successfully connected, speak into the microphone on one device and listen for audio on the other.

At this point, the audio signal should transfer to the headset. Listen to the headset to see if the data transfer is successful.

Step 5 (Optional) Click the **Disconnect** button on the Connection Status screen to close the connection.

The connection between Merlin's Wand and the device will terminate, and the **Connect** button will again be available. Selecting Connect will reestablish the connection.

Connect to Device: LAN

This section explains how to configure Merlin's Wand to establish an RFCOMM connection to a device that uses the Local Area Network Access profile.

Note To connect to a device that uses the LAN profile, you will need to install the virtual COM port driver that is included with the Merlin's Wand installation. For instructions, See "Installing the Virtual COM Port Driver" on page 39.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select LAN from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen.

Connect to Device: Object Push

Merlin's Wand can be configured to transfer files to a Bluetooth wireless device that complies with the Object Push profile. This section shows how to configure Merlin's Wand to connect and transfer files to a Bluetooth device that supports Object Push.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select Object Push from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Press **Next**.

The Select Data for Transfer screen will open:

Step 4 On the Select Data for Transfer screen, there are options to transfer a file or to transfer text.

- To transfer a file: Select the radio button next to "Transfer this file." Type in a filename or locate the file by clicking the **Browse** button to access the Open dialog. By default, the filename in the "Send data to the following file on the receiving device" box matches the name of the file to be transferred. If desired, enter a different filename in that box. When the desired file and target file's name have been entered, proceed to Step 7.

- To transfer text: Select the radio button next to "Transfer this text." Enter text in the text box. By default, the filename in the "Send data to the following file on the receiving device" box is "mw001.txt." If desired, enter a different target filename. When the desired text and target file's name have been entered, proceed to Step 7.

Step 5 Click **Next**.

The Transferring File screen will appear. First, a connection with the remote device will be established, and then data will begin transferring. A progress bar will show what percentage of the transfer has gone through. Click Stop Transfer to abort a transfer at any time.

When the transfer is complete, Merlin's Wand will disconnect from the target device:

At this point, you can click Back to transfer another file, or click Restart to start a new Profile Wizard session.

Connect to Device: PAN–GN

These steps explain how to use Profile Wizard to configure Merlin's Wand to connect to a Bluetooth device that supports the Personal Area Network–Group Ad-Hoc Network profile.

NoteTo connect to a device that uses the PAN profile, you must install the virtual network driver that is included with the Merlin's Wand installation. For instructions, see Section 3.7, "Installation of Network Driver," on page 41.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select PAN - GN from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen. Merlin's Wand will attempt to establish an ACL connection to the remote device, and then open a BNEP connection to it. Merlin's Wand will then send a Setup Connection Request Message. When Merlin's Wand receives the Setup Connection Response Message from the remote device, network services may be accessed.

Step 4 You can now perform the following operations:

- **Send Control Packet**: Use this to select a filter control message to send to the remote device. You may send either a Filter Network Protocol Type message or a Filter Multicast Address Type message control packet to the remote device.

Start Range and End Range: Start Range defines the beginning of the range, and End Range defines the end of the range. Only the protocol types or multicast addresses that fall within the range will be sent by the remote device; all other types or addresses will be filtered out.

For Filter Network Type Set Message, the Start and End Ranges may be in the format *nnnn* or 0x*nnnn*. For the Filter Multicast Address Set Message, the Start and End Ranges can be in the format *nnnnnnnnnnnn* or 0x*nnnnnnnnnnnn*.

- **Send Packet**: Use this to send an ethernet packet to the remote device. The types of ethernet packets that can be sent are General, Compressed, Compressed Source Only and Compressed Destination Only.

NoteDepending on your network settings for 'CATC Bluetooth Trainer PAN Virtual NIC' Local Connection, Windows may create additional network traffic over this connection. Check your network properties for 'CATC Bluetooth Trainer PAN virtual NIC.'

Connect to Device: PAN–NAP

Follow these steps to configure Merlin's Wand to connect to a Bluetooth device that supports the Personal Area Network–Network Access Point profile.

NoteTo connect to a device that uses the PAN profile, you must install the virtual network driver that is included with the Merlin's Wand installation. For instructions, see Section 3.7, "Installation of Network Driver," on page 41.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select PAN - NAP from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen. Merlin's Wand will attempt to establish an ACL connection to the remote device, and then open a BNEP connection to it. Merlin's Wand will then send a Setup Connection Request Message. When Merlin's Wand receives the Setup Connection Response Message from the remote device, network services may be accessed.

Step 4 You can now perform the following operations:

- **Send Control Packet**: Use this to select a filter control message to send to the remote device. You may send either a Filter Network Protocol Type message or a Filter Multicast Address Type message control packet to the remote device. The only types of network filtering allowed are ARP (0806) and IPv4 (0800).

  Start Range and End Range: Start Range defines the beginning of the range, and End Range defines the end of the range. Only the protocol types or multicast addresses that fall within the range will be sent by the remote device; all other types or addresses will be filtered out.

  For Filter Network Type Set Message, the Start and End Ranges may be in the format *nnnn* or 0x*nnnn*. For the Filter Multicast Address Set Message, the Start and End Ranges can be in the format *nnnnnnnnnnnn* or 0x*nnnnnnnnnnnn*.

- **Send Packet**: Use this to send an ethernet packet to the remote device. The types of ethernet packets that can be sent are General, Compressed, Compressed Source Only and Compressed Destination Only.

Note Depending on your network settings for 'CATC Bluetooth Trainer PAN Virtual NIC' Local Connection, Windows may create additional network traffic over this connection. Check your network properties for 'CATC Bluetooth Trainer PAN virtual NIC.'

Connect to Device: Serial Port

These steps describe how to use Profile Wizard to configure Merlin's Wand to connect to a device that uses the Serial Port profile for serial port emulation.

Note To connect to a device that uses the Serial Port profile, you will need to install the virtual COM port driver that is included with the Merlin's Wand installation. For instructions, see "Installing the Virtual COM Port Driver" on page 39.

Step 1 Complete steps 1-6 in Section 3.2, "Connecting to Devices" on page 17.

Step 2 Select Serial Port from the list on the Select Profile screen.

(Optional) Select the Allow Role Switch option on the screen to enable Merlin's Wand to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen, and Merlin's Wand will attempt to establish a connection to the device.

If the connection attempt is successful, the Connection Status screen will show that Merlin's Wand has established a connection with the device.

Note If you cannot establish a connection, you can re-attempt the connection by either pressing Back and re-running the previous two steps, or by pressing the **Connect** button on the Connection Status screen.

Step 4 Use an external application to send data to the remote device.

Note A modem driver must be installed on the virtual COM port in order to use an external application to send data through Merlin's Wand. See "Install a Modem Driver on the Virtual COM Port" on page 40 to find out how to install the driver. Once the driver is installed, you must configure the external application to use that driver.

Open the application and send the data. It will use the modem driver on the virtual COM port to send the data to the Serial Port device through Merlin's Wand.

3.3 Emulating Devices

Choosing the "Emulate Device" option on the opening screen of Profile Wizard allows you to configure Merlin's Wand to emulate a Bluetooth wireless device. This option causes Merlin's Wand to act as a slave device.

Step 1 Open Profile Wizard and click the **Emulate Device** button.

The Select Profile screen will open.

Step 2 Select a profile from the list and click **Next**.

You will advance to the Configure Security screen.

Step 3 Select the security option that you would like to apply for the device.

- *Do not apply authentication and encryption*.
  Authentication and encryption will not be used.
- *Apply authentication and encryption*.
  The default authentication and encryption settings will be used.
- *Use current authentication/encryption settings*.
  The authentication and encryption settings that are currently set in the Local Device manager will be used.
- *Display the Local Device Manager so I can configure security and verify PIN codes myself.*
  This option allows you to access the Local Device Manager to

manually enter the security settings. If you choose this option, press Next to open the Local Device Manager. When you close the dialog, Step 4 will be performed automatically.

Step 4 Press **Next**.

At this point, please refer to the section that describes the setup for the particular profile that the target device uses:

- Section , “Emulate Device: Dial-Up Gateway” on page 31
- Section , “Emulate Device: Fax Gateway” on page 32
- Section , “Emulate Device: File Transfer” on page 33
- Section , “Emulate Device: HCRP Server” on page 33
- Section , “Emulate Device: Headset” on page 34
- Section , “Emulate Device: Headset Audio Gateway” on page 35
- Section , “Emulate Device: LAN” on page 35
- Section , “Emulate Device: Object Push” on page 36
- Section , “Emulate Device: PAN - GN” on page 36
- Section , “Emulate Device: PAN - NAP” on page 37
- Section , “Emulate Device: Serial Port” on page 38

Emulate Device: Dial-Up Gateway

These steps show how to use Profile Wizard to configure Merlin's Wand to emulate a dial-up gateway device.

Step 1 Complete Steps 1-4 in Section 3.3, “Emulating Devices” on page 30.

The Select Dial-Up Emulation screen will open.

Step 2 You may choose to configure Merlin's Wand to emulate a dial-up gateway device while connected to or emulating a modem or while connected to a virtual COM port.

To connect to a modem:

(a) Select “Connect to modem” and choose a modem from the list.

(b) Press **Next**.
At this point, initialize a connection to Merlin's Wand from a Bluetooth device that uses the Dial-Up Gateway profile. When the connection has been established, the modem status can be observed via the status lights.

(c) (Optional) Enable or disable event logging:
You can configure Merlin's Wand to write incoming or outgoing data to the Event Log by checking or unchecking the “Log Incoming Data” and “Log Outgoing Data” options.

To emulate a modem:

(a) Select "Emulate modem."

(b) Press **Next**.
At this point, initialize a connection to Merlin's Wand from a Bluetooth device that uses the Dial-Up Gateway profile. When the connection has been established, the modem status can be observed via the status lights.

(c) Use the Response combo box to manually send modem responses to the device. You can choose a response from the drop-down list or enter them manually. Check "Automatically send response when cmd is received" so that the currently highlighted response will be sent automatically.

To connect to virtual COM port:

NoteTo connect to a virtual COM port, you will need to install the virtual COM port driver that is included with the Merlin's Wand installation. For instructions, see "Installing the Virtual COM Port Driver" on page 39.

(a) Select "Connect to virtual COM port."

(b) Press **Next**.
At this point, initialize a connection to Merlin's Wand from a Bluetooth device that uses the Dial-Up Gateway profile. When the connection has been established, you can use an external application to communicate with the device via Merlin's Wand.

Emulate Device: Fax Gateway

The following steps show how to configure Merlin's Wand to emulate a fax gateway device.

Step 1 Complete Steps 1-4 in Section 3.3, "Emulating Devices" on page 30.

The select Fax Emulation screen will open.

Step 2 You may choose between configuring Merlin's Wand to emulate a fax gateway device while connected to a modem or while connected to a virtual COM port.

To connect to a modem:

(a) Select "Connect to modem" and choose a modem from the list.

(b) Press **Next**.
At this point, initialize a connection to Merlin's Wand from a Bluetooth device that uses the Fax Gateway profile. When the connection has been established, the modem status can be observed via the status lights.

(c) (Optional) Enable or disable event logging:
You can configure Merlin's Wand to write incoming or outgoing data to the Event Log by checking or unchecking the "Log Incoming Data" and "Log Outgoing Data" options.

To connect to virtual COM port:

NoteTo connect to a virtual COM port, you will need to install the virtual COM port driver that is included with the Merlin's Wand installation. For instructions, see "Installing the Virtual COM Port Driver" on page 39.

(a) Select "Connect to virtual COM port" and choose the fax classes that Merlin's Wand will support during emulation.

(b) Press **Next**.
At this point, initialize a connection to Merlin's Wand from a Bluetooth device that uses the Fax Gateway profile. When the connection has been established, you can use an external application to communicate with the device via Merlin's Wand.

Emulate Device: File Transfer

Follow these steps to configure Merlin's Wand to emulate a Bluetooth device that supports the File Transfer profile.

Step 1 Complete Steps 1-4 in Section 3.3, "Emulating Devices" on page 30.

The File Transfer Emulation screen will open.

Step 2 (Optional) Use the **Change...** button to change the root folder that the remote device will access.

Step 3 Direct the remote device to connect to Merlin's Wand.

An OBEX connection will be established between the device and Merlin's Wand.

Step 4 At this point, you may transfer files to the root folder from the remote device.

Emulate Device: HCRP Server

This section explains how to use Profile Wizard to set up Merlin's Wand to emulate an HCRP server.

Step 1 Complete Steps 1-4 in Section 3.3, "Emulating Devices" on page 30.

The Printer Emulation screen will open.

Step 2 You can now do any of the following:

- Change the folder in which transferred files are stored: Click the **Change...** button to open the Browse for Folder dialog. Select the folder in which you want transferred files to be stored. Click **OK** to confirm the change.
- Change the maximum size for data credits: Click the Change maximum size... button to open the Change Data Credit Maximum Size dialog. Type in a new credit size, in bytes. Click **OK** to confirm the change.
- Set the LPT status bits: Check or uncheck the status bits to change their settings.

  Paper empty: Checked = 1 = Paper empty; Unchecked = 0 = Paper not empty

  Selected: Checked = 1 = Selected; Unchecked = 0 = Not selected

  Error: Checked = 0 = Error; Unchecked = 1 = No error
- Change or add a printer ID string: Click **Change printer ID string...** to open the Change Printer ID String dialog. Enter strings as KEY:VALUE pairs or vendor-specific pairs. Individual strings must be separated by semicolons. Click **OK** to confirm changes.
- Send a file to the printer: Click the **Print...** button to access the Open dialog. By default, the file that was most recently transferred to the HCRP server will be selected; however, you may browse to a different file, if you wish. Click **Open** to print the file.

Emulate Device: Headset

Merlin's Wand can be configured to emulate a wireless device that conforms to the Bluetooth Headset profile. The following steps show how to set up Merlin's Wand as a Headset device and connect to it with a remote Bluetooth headset.

Step 1 Complete Steps 1-4 in Section 3.3, "Emulating Devices" on page 30.

The Headset Emulation screen will open, indicating that Merlin's Wand has been configured to emulate a device that supports the Headset profile and is awaiting connection from a device.

NoteThe Speaker and Microphone Volume levels shown in the previous screenshot reflect the volume settings on the Master device. If you adjust the levels on the remote device, the displayed volume levels will change accordingly.

Step 2 Direct a remote Bluetooth device to connect to Merlin's

Wand.

Once the connection is established, the Emulation Status screen will indicate that Merlin's Wand has an RFCOMM connection to the device.

Step 3 Click the **Answer** button to make an SCO connection with the remote device.

If the connection attempt is succesful, the screen will change to indicate that an SCO connection has been established.

Step 4 (Optional) Click the **Hang Up** button to close the connection. The connection between Merlin's Wand and the device will terminate.

Emulate Device: Headset Audio Gateway

Merlin's Wand can be configured to emulate a wireless device that conforms to the Bluetooth Headset Audio Gateway profile. The following steps show how to set up Merlin's Wand as a Headset Audio Gateway device and connect to it with a remote Bluetooth headset.

Step 1 Complete Steps 1-4 in Section 3.3, "Emulating Devices" on page 30.

The Headset AG Emulation screen will open, indicating that Merlin's Wand has been configured to emulate a device that supports the Headset Audio Gateway profile and is awaiting connection from a device.

NoteThe Speaker and Microphone Volume levels shown in the previous screenshot reflect the volume settings on the Master device. If you adjust the levels on the remote device, the displayed volume levels will change accordingly.

Step 2 Direct a remote Bluetooth device to connect to Merlin's Wand.

Once the connection is established, the Emulation Status screen will indicate that Merlin's Wand is currently connected to the device.

Step 3 To verify that Merlin's Wand and the remote device are successfully connected, speak into the microphone on one device and listen for audio on the other.

Emulate Device: LAN

Step 1 Complete Steps 1-4 in Section 3.3, "Emulating Devices" on page 30.

Step 2 In the Emulate Device dialog, select LAN.

The LAN screen will open, indicating that Merlin's Wand has been configured to emulate a device that supports the LAN profile and is awaiting connection from a device.

Emulate Device: Object Push

Merlin's Wand can emulate the file transfer capabilities of wireless devices that support the Object Push profile through the Object Push option. Object Push emulation allows other devices to transfer files to Merlin's Wand.

Step 1 Complete Steps 1-4 in Section 3.3, "Emulating Devices" on page 30.

The Emulation Status screen will open, indicating that Merlin's Wand has been configured to emulate a Bluetooth device that supports the Object Push profile and is awaiting connection from a device. It is now ready to receive files.

Step 2 If desired, the folder in which transferred files are stored can be changed. To change it, click the **Change** button and select a new directory in the Browse for Folder dialog.

Step 3 Initiate file transfer from the Bluetooth device.

The Emulation Status screen will show the file transfer progress:

When the transfer is complete, the Emulation Status screen will show that Merlin's Wand is waiting for a connection after having successfully received the file.

Emulate Device: PAN - GN

Profile Wizard allows you to emulate devices that use the Personal Area Network (PAN)- General Networking (GN) profile.

NoteTo connect to a device that uses the PAN profile, you must install the virtual network driver that is included with the Merlin's Wand installation. For instructions, see Section 3.7, "Installation of Network Driver," on page 41.

Step 1 Complete Steps 1-4 in Section 3.3, "Emulating Devices" on page 30.

Step 2 In the Select Profile screen, select PAN-GN, and click Next.

The Emulation Status screen will open, indicating that Merlin's Wand has been configured to emulate a Bluetooth device that supports the PAN-GN profile and is awaiting connection from a device. It is now ready to receive files.

Step 3 Click Next.

Step 4 Select the appropriate security options, and click Next.

- **Send Control Packet**: Use this to select a filter control message to send to the remote device. You may send either a Filter Network Protocol Type message or a Filter Multicast Address Type message control packet to the remote device.

  Start Range and End Range: Start Range defines the beginning of the range, and End Range defines the end of the range. Only the protocol types or multicast addresses that fall within the range will be sent by the remote device; all other types or addresses will be filtered out.

  For Filter Network Type Set Message, the Start and End Ranges may be in the format *nnnn* or 0x*nnnn*. For the Filter Multicast Address Set Message, the Start and End Ranges can be in the format *nnnnnnnnnnnn* or 0x*nnnnnnnnnnnn*.

- **Send Packet**: Use this to send an ethernet packet to the remote device. The types of ethernet packets that can be sent are General, Compressed, Compressed Source Only and Compressed Destination Only.

NoteDepending on your network settings for 'CATC Bluetooth Trainer PAN Virtual NIC' Local Connection, Windows may create additional network traffic over this connection. Check your network properties for 'CATC Bluetooth Trainer PAN virtual NIC.'

Emulate Device: PAN - NAP

NoteTo connect to a device that uses the PAN profile, you must install the virtual network driver that is included with the Merlin's Wand installation. For instructions, see Section 3.7, "Installation of Network Driver," on page 41.

Step 1 Complete Steps 1-4 in Section 3.3, "Emulating Devices" on page 30.

Step 2 In the Select Profile screen, select PAN-NAP, and click Next.

The Emulation Status screen will open, indicating that Merlin's Wand has been configured to emulate a Bluetooth device that supports the PAN-NAP profile and is awaiting connection from a device. It is now ready to receive files.

Step 3 Click Next.

- **Send Control Packet**: Use this to select a filter control message to send to the remote device. You may send either a Filter Network Protocol Type message or a Filter Multicast Address Type message control packet to the remote device.

  Start Range and End Range: Start Range defines the beginning of the range, and End Range defines the end of the range. Only the protocol types or multicast addresses that fall within the range will be sent by the remote device; all other types or addresses will be filtered out.

  For Filter Network Type Set Message, the Start and End Ranges may be in the format *nnnn* or 0x*nnnn*. For the Filter Multicast Address Set Message, the Start and End Ranges can be in the format *nnnnnnnnnnnn* or 0x*nnnnnnnnnnnn*.
- **Send Packet**: Use this to send an ethernet packet to the remote device. The types of ethernet packets that can be sent are General, Compressed, Compressed Source Only and Compressed Destination Only.

NoteDepending on your network settings for 'CATC Bluetooth Trainer PAN Virtual NIC' Local Connection, Windows may create additional network traffic over this connection. Check your network properties for 'CATC Bluetooth Trainer PAN virtual NIC.'

Emulate Device: Serial Port

These steps describe how to use Profile Wizard to configure Merlin's Wand to emulate a device that uses the Serial Port profile.

NoteTo emulate a device that uses the Serial Port profile, you will need to install the virtual COM port driver that is included with the Merlin's Wand installation. For instructions, see "Installing the Virtual COM Port Driver" on page 39.

Step 1 Complete Steps 1-4 in Section 3.3, "Emulating Devices" on page 30.

The Serial Port Emulation screen will open.

Step 2 Use an external application to send data to the remote device.

NoteA modem driver must be installed on the virtual COM port in order to use an external application to send data through Merlin's Wand. See "Install a Modem Driver on the Virtual COM Port" on page 40 to find out how to install the driver. Once the driver is installed, you must configure the external application to use that driver.

Open the application and send the data. It will use the modem driver on the virtual COM port to send the data to the Serial Port device through Merlin's Wand.

3.4 Restarting the Wizard

When working within Profile Wizard, a new session may be started at any time.

Click the **Restart Wizard** button on any screen.

- If Merlin's Wand is currently emulating a connected device, the connection will be broken, and Profile Wizard will return to the default Profile Wizard screen.
- If Merlin's Wand is currently connected to a device, a dialog box will ask if the connection should be terminated. Clicking the **Yes** button will cause the connection to be broken, and Profile Wizard will return to the default Profile Wizard screen. Clicking the No button will cancel the Restart Wizard request.

3.5 Installing the Virtual COM Port Driver

To connect to a device that uses the Dial-Up Gateway, Fax Gateway, LAN, or Serial Port profile, you will need to install the virtual COM port driver that is included with the Merlin's Wand installation.

On Windows 2000

Step 1 Select Start > Settings > Control Panel from the desktop taskbar, then double-click on Add/Remove Hardware in the Control Panel window.

The Add/Remove Hardware Wizard will open.

Step 2 Click **Next**.

Step 3 Select "Add/Troubleshoot a device" and click Next.

The Wizard will look for Plug and Play hardware, then it will display a list of all the hardware it finds on the computer.

Step 4 Select "Add a new device" from the list and click Next.

Step 5 Select "No, I want to select the hardware from a list" and click Next.

Step 6 Select "Ports (COM and LPT)" from the list and click Next.

Step 7 Click the **Have Disk** button.

Step 8 Click the **Browse** button so that you can browse to the driver file, which you should be able to find either on the installation CD-ROM or in the directory in which Merlin's Wand is installed.

When you have located the file, click **Open**.

Step 9 Click **OK** to use the file you have selected.

Step 10 Select "CATC Bluetooth Serial Port" and click **OK**.

Step 11 Click **Next** to install the driver.

Step 12 Click **Finish** to complete the installation.

Ascertain the COM Port Number

Step 1 Select Start > Settings > Control Panel from the desktop taskbar, then double-click on System in the Control Panel window.

The System Properties dialog will open.

Step 2 Select the Hardware tab and click the **Device Manager** button.

The Device Manager will open.

Step 3 Expand the Ports (COM & LPT) level and locate the CATC Bluetooth Serial Port.

Step 4 Note the port number for the CATC Bluetooth Serial Port. You will need to know the number in order to install things such as the fax modem driver that is necessary to connect to a fax device.

3.6 Install a Modem Driver on the Virtual COM Port

In order to perform dial-up networking or use a fax application to send fax data through Merlin's Wand, you need to install a modem driver on the virtual COM port.

On Windows 2000:

Step 1 Complete steps 1-3 in Section , "Connect to Device: Dial-Up Gateway" on page 19 or in Section on page 21.

Merlin's Wand must be connected to the Dial-Up or Fax Gateway device before installing the modem driver.

Step 2 Select **Start** > **Settings** > **Control Panel** from the desktop taskbar, then double-click on **Add/Remove Hardware** in the Control Panel window.

The Add/Remove Hardware Wizard will open.

Step 3 Click **Next**.

Step 4 Select "Add/Troubleshoot a device" and click **Next**.

The Wizard will look for Plug and Play hardware, then it will display a list of all the hardware it finds on the computer.

Step 5 Select "Add a new device" from the list and click **Next**.

Step 6 Select "No, I want to select the hardware from a list" and click **Next**.

Step 7 Select "Modems" from the list and click **Next**.

Step 8 Check "Don't detect my modem; I will select it from a list." and click **Next**.

Step 9 Select the manufacturer and model of your modem and click **Next**.

Step 10 Select the virtual COM port from the list and click Next.

NoteTo find out the number of the virtual COM port, refer to "Ascertain the COM Port Number" on page 40.

Step 11 Click the **Finish** button to complete the installation.

3.7 Installation of Network Driver

BTTNet.sys driver + INF

Step 1 Open 'Control Panel'

Step 2 Select 'Add/Remove Hardware' icon.

Step 3 Select the 'Add/Troubleshoot a device' radio button. Pressing **Next** would make the system try to search for new HW devices.

Step 4 In the Device List, select "Add a new device" entry. Press **Next** button.

Step 5 Select the "No, I want to select the hardware from a list" radio button. Press **Next**.

Step 6 Select "Network Adapters" entry. Press **Next**.

Step 7 In the 'Select Network Adapter' page, press "**Have Disk ...**"

button.

Step 8 Browse for the BTTNet.INF file and select it by pressing the "**Open**" and "**OK**" buttons. The list of network adapters should include the driver "CATC Bluetooth Trainer PAN virtual NIC."

Step 9 Press **Next**.

Step 10 In the "Start Hardware Installation" page, press the **Next** button. At this time, Windows installs the driver and plots a proper text in the installation wizard.

Step 11 Press **Finish**.

Verifying the Driver's Installation

To verify that the drivers installed properly, go to the "Device Manager" utility, and display the properties for the "CATC Bluetooth Trainer PAN virtual NIC" driver under "Network Adapters."

NoteThe MAC address of the NIC driver is based on the BD address of Merlin's Wand. The MAC addr will be rewritten every time the BNEP_Register is executed.

Configuration of Virtual NIC

To configure the network properties of the "CATC Bluetooth Trainer PAN Virtual NIC," do the following:

Step 1 Select **Start** > **Settings** > **Control Panel** from the desktop taskbar, then double-click on "Network and Dial-up Connections" in the Control Panel window.

Step 2 Locate the "CATC Bluetooth Trainer PAN virtual NIC" Local Area Connection.

Step 3 Select **File** > **Properties**. Now you can change the network settings for your Bluetooth network connection.

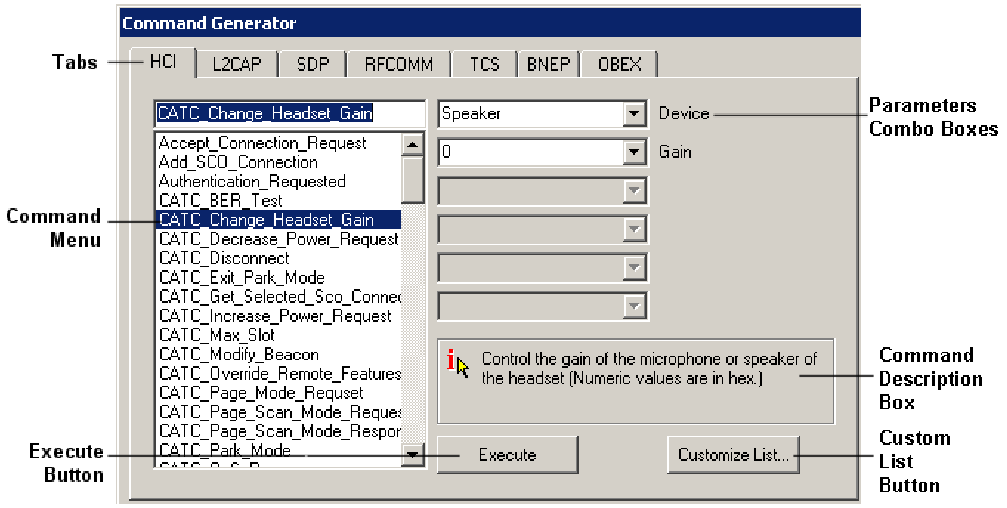
4. Command Generator

The Command Generator is a tool in Merlin's Wand that presents a menu of protocol commands so that you can select and execute any command in virtually any sequence. Command Generator thus gives maximum control over the traffic generation process.

Command Generator requires that you build connections from the Baseband level on up. This means that to establish an OBEX connection, for example, you will need to first start with Baseband and work your way up the protocol stack. You cannot simply start at a higher protocol.

The utility displays a window with tabs for these protocols: HCI (which induces Baseband, LMP and Module-Specific Commands), L2CAP, SDP, RFCOMM, TCS, and OBEX. Clicking a tab or a name in the protocol stack graphic opens a window and presents a menu of commands for that protocol.

4.1 Layout of the Command Generator



CATC_Change_Headset_Gain

Device: Speaker

Gain: 0

Control the gain of the microphone or speaker of the headset (Numeric values are in hex.)

The Command Generator utility is composed of the following:

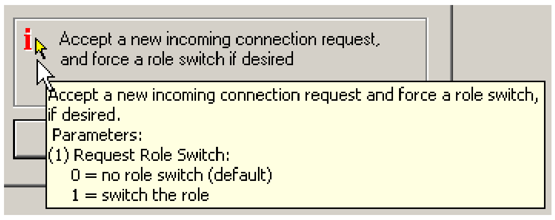
Tabs -- There is a tab for each of six protocols: HCI, L2CAP, SDP, RFCOMM, TCS, and OBEX. Clicking a tab displays the Command Menu for the chosen protocol.

Command Menu -- A list of commands is provided for each protocol.

Parameters Combo Boxes -- Parameters can be entered via the six combo boxes. One or more of the boxes may be activated, depending on which command is currently selected in the Command Menu. Parameters may either be typed into the box or chosen from a pull-down list within the box.

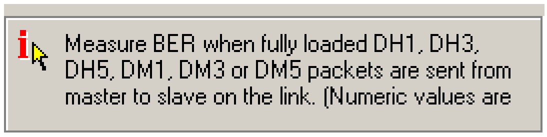
Execute -- Pressing the Execute button Execute will cause Merlin's Wand to run the selected command.

Command Generator Tips -- Detailed tips for each command are accessible by positioning the mouse over the question mark icon. A pop-up window that contains detailed information about the selected command will appear.


Accept a new incoming connection request, and force a role switch if desired

Accept a new incoming connection request and force a role switch, if desired.
Parameters:
(1) Request Role Switch:
0 = no role switch (default)
1 = switch the role

Command Description Box -- A short description will display in the Command Description Box when a command is selected from a Command Menu.


Measure BER when fully loaded DH1, DH3, DH5, DM1, DM3 or DM5 packets are sent from master to slave on the link. (Numeric values are

Protocol Stack Graphic -- At the bottom of Command Generator is the Protocol Stack Graphic, which illustrates the layers that make up the Bluetooth protocol stack. The protocols in the graphic are also clickable buttons that can be used to access the command menus for each protocol.

HCI Customized List Button -- The HCI tab has an additional button >> to the left of the Execute button. It provides access to an interface that allows the user to customize the list of commands displayed in the HCI command menu.

4.2 Using Command Generator

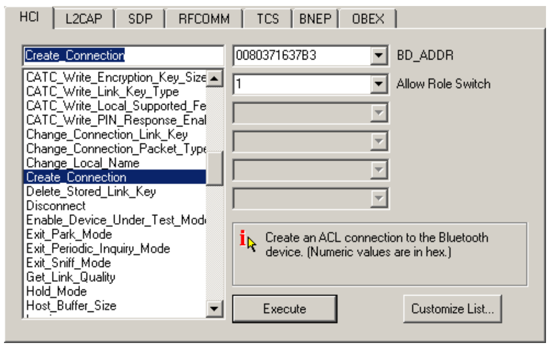
Note: If Command Generator isn't enabled on your Merlin's Wand system, you will need to obtain a License Key from CATC before you can use it. See "Several frequently-used operations are bound to keyboard shortcuts." on page 15 for details.

To execute commands with Command Generator:

Step 1 Click the Command Generator button on the toolbar or select the command **Tools > Command Generator** from the menu bar.

The Command Generator utility will open.

Step 2 **Choose a protocol** to work with by clicking one of the five tabs or a layer in the Protocol Stack Graphic.



Create an ACL connection to the Bluetooth device. (Numeric values are in hex.)

The list of available commands for the chosen protocol will display in the Command Menu.

Note: The HCI tab is displayed by default.

Step 3 **Select a command** from the Command Menu.

A description of the command will display in the Command Description Box. If the selected command is not supported, the message in the Command Description Box will read "Not supported."

Step 4 **Enter parameters**, if required, in the Parameters Combo Boxes. Parameter boxes will be activated as appropriate for the command, with the parameter name(s) appearing to the right of the box(es). Data can be typed directly into the Parameters Combo Boxes, and some of the boxes may offer drop-down lists from which to select the appropriate parameter.

Note: Numeric values should be entered as hexadecimal unless otherwise specified.

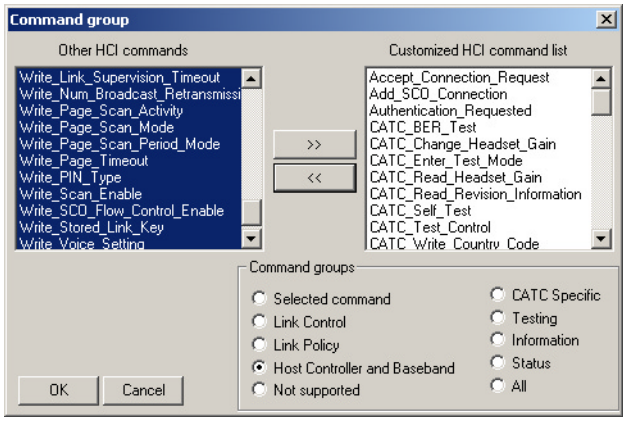
Step 5 Click the **Execute** button to run the command.

Note: While Command Generator offers maximum control over Merlin's Wand, there are times when command choices may be limited. Some lower-level connections may prevent access to commands for higher-level protocols. For example, if an L2CAP connection has been established between Merlin's Wand and a device, it is not possible to work with OBEX commands in Command Generator. Merlin's Wand will display a message to indicate that L2CAP connections must be closed before working with OBEX commands. Once the L2CAP connection is closed, the OBEX commands will be accessible.

Customizing the List of HCI Commands

The list of commands in the HCI command menu in Command Generator may be customized to display only certain commands. Since there are over 100 commands available in the HCI menu, this feature is a handy way to eliminate scrolling through a lengthy list to find commands.

Clicking the HCI Customized List button >>, which is located to the left of the Execute button in Command Generator, will open the Command Group interface.



Other HCI commands

Write_Link_Supervision_Timeout
Write_Num_Broadcast_Retransmissi
Write_Page_Scan_Activity
Write_Page_Scan_Mode
Write_Page_Scan_Period_Mode
Write_Page_Timeout
Write_PIN_Type
Write_Scan_Enable
Write_SCO_Flow_Control_Enable
Write_Stored_Link_Key
Write_Voice_Setting

Customized HCI command list

Accept_Connection_Request
Add_SCO_Connection
Authentication_Requested
CATC_BER_Test
CATC_Change_Headset_Gain
CATC_Enter_Test_Mode
CATC_Read_Headset_Gain
CATC_Read_Revision_Information
CATC_Self_Test
CATC_Test_Control
CATC_Write_Country_Code

Command groups

- ○ Selected command
- ○ Link Control
- ○ Link Policy
- ● Host Controller and Baseband
- ○ Not supported
- ○ CATC Specific
- ○ Testing
- ○ Information
- ○ Status
- ○ All

To remove commands from the customized HCI command list, select the radio button beside one of the groups listed in the "Command groups" section of the interface, then press the Remove button <<. The selected command(s) will move into the "Other HCI commands" list.

To add commands to the customized list, select the radio button next to the group of commands that should be moved, then press the Add button >>. The selected command(s) will be moved from "Other HCI commands" to the customized HCI command list.

4.3 Tables of Available Commands

The following tables summarize the commands in Command Generator.

For detailed descriptions of the commands, see *Appendix A: Command Generator Command Descriptions*, on page 87.

Note **"N/A"** means Not Applicable. This indicates that the specified command does not have a parameter.

HCI Commands

Link Control Commands

Two sections of Link Control Commands are presented. The first section lists commands that are supported by Merlin's Wand. The second section presents commands that are *not* supported.

Supported

Commands	Parameters
Accept_Connection_Request	N/A
Add_SCO_Connection	HCI_Handle Packet Type
Authentication_Requested	HCI_Handle
Change_Connection_Link_Key	HCI_Handle
Change_Connection_Packet_Type	HCI_Handle Packet_Type
Create_Connection	BD_ADDR
Disconnect	HCI_Handle
Exit_Periodic_Inquiry_Mode	N/A
Inquiry	Inquiry_Length Num_Responses
Inquiry_Cancel	N/A
Periodic_Inquiry_Mode	Max Period Length Min Period Length Inquiry Length Num of Responses
PIN_Code_Request_Negative_Reply	BD_ADDR
PIN_Code_Request_Reply	PIN Code BD_ADDR
Read_Clock_Offset	HCI_Handle
Read_Remote_Supported_Features	HCI_Handle
Read_Remote_Version_Information	HCI_Handle
Reject_Connection_Request	N/A
Remote_Name_Request	BD ADDR Page Scan Rep Mode Page Scan Mode Clock Offset
Set_Connection_Encryption	HCI_Handle Encryption_Enable

Not supported

Commands
Link_Key_Request_Negative_Reply
Link_Key_Request_Reply
Master_Link_Key

Link Policy Commands

Supported

Commands	Parameters
Exit_Park_Mode	HCI_Handle
Exit_Sniff_Mode	HCI_Handle
Hold_Mode	HCI_Handle Max_Interval Min_Interval
Park_Mode	HCI_Handle Beacon_Max_Interval Beacon_Min_Interval
QoS_Setup	HCI_Handle ServiceType TokenRate PeakBandwidth Latency DelayVariation
Read_Link_Policy_Settings	HCI_Handle
Role_Discovery	HCI_Handle
Sniff_Mode	HCI_Handle Max_Interval Min_Interval Attempt Timeout
Switch_Role	BD_ADDR
Write_Link_Policy_Settings	HCI_Handle Link_Policy_Settings

Not supported

All Link Policy commands are supported in Command Generator.

Host Controller & Baseband Commands

Supported

Commands	Parameters
Change_Local_Name	Name
Delete_Stored_Link_Key	BD_ADDR Delete_All_Flag
Host_Buffer_Size	ACL_Data_Length SCO_Data_Length Total_Num_ACL Total_Num_SCO
Read_Authentication_Enable	N/A

Commands	Parameters
Read_Class_of_Device	N/A
Read_Connection_Accept_Timeout	N/A
Read_Current_IAC_LAP	N/A
Read_Encryption_Mode	N/A
Read_Local_Name	N/A
Read_Link_Supervision_Timeout	HCI_Handle
Read_Number_Of_Supported_IAC	N/A
Read_Page_Scan_Mode	N/A
Read_Page_Scan_Period_Mode	N/A
Read_Page_Timeout	N/A
Read_PIN_Type	N/A
Read_Scan_Enable	N/A
Read_SCO_Flow_Control_Enable	N/A
Read_Stored_Link_Key	BD_ADDR Read_All_Flag
Read_Voice_Setting	N/A
Reset	N/A
Set_Event_Filter	FilterType FilterConditionType Condition
Set_Event_Mask	Event_Mask
Write_Authentication_Enable	Authentication_Enable
Write_Class_of_Device	CoD
Write_Connection_Accept_Timeout	Timeout
Write_Current_IAC_LAP	IAC_LAP
Write_Encryption_Mode	Encryption Mode
Write_Link_Supervision_Timeout	HCI_Handle Timeout
Write_Page_Timeout	Timeout
Write_PIN_Type	PIN_Type
Write_Scan_Enable	Scan_Enable
Write_Stored_Link_Key	BD_ADDR Link_Key
Write_Voice_Settings	HCI_Handle Voice_Setting

Not Supported

Commands
Create_New_Unit_Key
Flush
Host_Number_Of_Completed_Packets
Read_Automatic_Flush_Timeout
Read_Hold_Mode_Activity
Read_Inquiry_Scan_Activity
Read_Num_Broadcast_Retransmissions
Read_Page_Scan_Activity
Read_Transmit_Power_Level
Set_Host_Controller_To_Host_Flow_Control
Write_Automatic_Flush_Timeout
Write_Hold_Mode_Activity
Write_Inquiry_Scan_Activity
Write_Num_Broadcast_Retransmissions
Write_Page_Scan_Activity
Write_Page_Scan_Mode
Write_Page_Scan_Period_Mode
Write_SCO_Flow_Control_Enable

Informational Commands

Supported

Commands	Parameters
Read_BD_ADDR	N/A
Read_Buffer_Size	N/A
Read_Country_Code	N/A
Read_Local_Supported_Features	N/A
Read_Local_Version_Information	N/A

Not Supported

There are no unsupported Informational commands.

Status Commands

Supported

There are no supported Status commands.

Not Supported

Commands
Get_Link_Quality
Read_Failed_Contact_Counter
Reset_Failed_Contact_Counter
Read_RSSI

Testing Commands

Supported

Commands	Parameters
Enable_Device_Under_Test_Mode	N/A
Read_Loopback_Mode	N/A
Write_Loopback_Mode	Loopback_Mode

Not Supported

There are no unsupported Testing commands.

CATC-Specific Commands

Supported

Commands	Parameters
CATC_BER	HCI_Handle Number_Of_Packets BER_Packet_Type Test_Data_Type Test_Data BER_Interval
CATC_Change_Headset_Gain	Device Gain
CATC_Read_Headset_Gain	Device
CATC_Read_PIN_Response_Enable	N/A
CATC_Read_Revision_Information	N/A
CATC_Self_Test	N/A
CATC_Set_Default_PIN_Code	Pin_Code
CATC_Write_PIN_Response_Enable	PIN-Response_Enable

L2CAP Commands

Supported

Commands	Parameters
ConfigurationResponse	Reason
ConfigurationSetup	ServiceType TokenRate TokenBucketSize PeakBandWidth Latency DelayVariation
ConnectRequest	HCI_Handle PSM Receive MTU
ConnectResponse	Response
DeregisterPsm	PSM
DisconnectRequest	CID
EchoRequest	HCI_Handle Data
InfoRequest	HCI_Handle
RegisterPsm	PSM Receive MTU
SendData	CID Data Pipe

Not Supported

Commands
GetRegisteredGroups
GroupDestroy
GroupRegister

SDP Commands

Supported

Commands	Parameters
AddProfileServiceRecord	Profile ServerChannel
AddServiceRecord	Filename Record Name Server Channel
ProfileServiceSearch	HCI_Handle Profile

Commands	Parameters
RequestServiceAttribute	HCI_Handle ServiceRecordHandle AttributeID AttributeID AttributeID
RequestServiceSearch	HCI_Handle ServiceClassID ServiceClassID ServiceClassID
RequestServiceSearchAttribute	HCI_Handle ServiceClassID ServiceClassID ServiceClassID
ResetDatabase	N/A

RFCOMM Commands

Supported

Commands	Parameters
AcceptChannel	Accept
AcceptPortSettings	Accept
AdvanceCredit	(HCI/DLCI) Credit
CloseClientChannel	(HCI/DLCI)
CreditFlowEnabled	(HCI/DLCI)
DeregisterServerChannel	ServerChannel
OpenClientChannel	HCI_Handle ServerChannel MaxFrameSize Credit
RegisterServerChannel	N/A
RequestPortSettings	(HCI/DLCI) BaudRate DataFormat FlowControl Xon Xoff
RequestPortStatus	(HCI/DLCI)
SendATCommand	(HCI/DLCI) AT_Command
SendData	(HCI/DLCI) Data Pipe

Commands	Parameters
SendTest	(HCI/DLCI)
SetLineStatus	(HCI/DLCI) LineStatus
SetModemStatus	(HCI/DLCI) ModemSignals Break Length

TCS Commands

Supported

Commands	Parameters
Register_Intercom_Profile	N/A
Open_TCS_Channel	HCI_Handle
Start_TCS_Call	N/A
Disconnect_TCS_Call	N/A
Send_Info_Message	Phone_Number

Not Supported

Current TCS implementation in Merlin's Wand supports only the Intercom profile. The Cordless profile that uses TCS is not currently supported by Merlin's Wand.

OBEX Commands

Supported

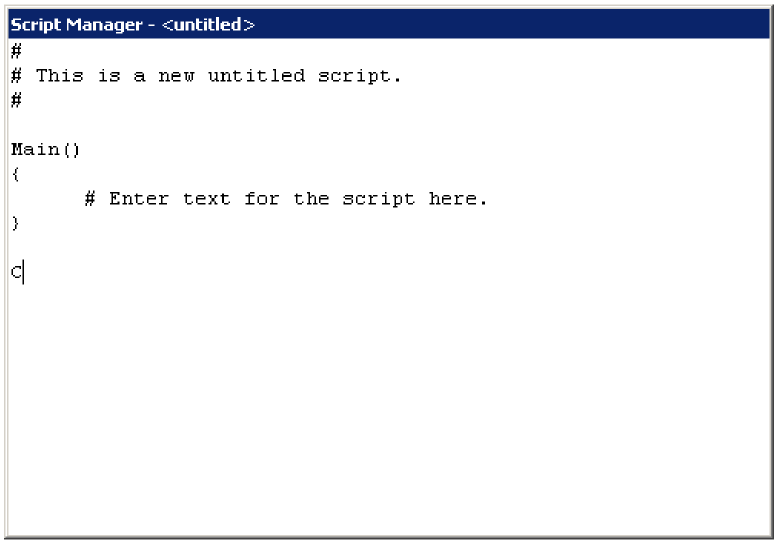
Commands	Parameters
ClientConnect	BD_ADDR
ClientDisconnect	N/A
ClientGet	Object
ClientPut	Filename
ClientSetPath	Path Flags
ServerDeinit	N/A
ServerInit	N/A
ServerSetPath	Path
ClientAbort	N/A

5. Script Manager

Script Manager is a tool within Merlin's Wand that presents a text editor window for writing and executing scripts. Scripts can be used to automate Bluetooth command sequences, making the testing process more efficient.

This chapter introduces the Script Manager interface. There are a number of commands available to you for writing scripts in Merlin's Wand. Command descriptions can be found in *Appendix C: Merlin's Wand Scripting Commands*, on page 155.

5.1 Layout of the Script Manager Window



```
#
# This is a new untitled script.
#

Main()
{
	# Enter text for the script here.
}

c
```

The Script Manager is essentially a text editor utility. You can create, save, print, edit, and open scripts in this window. Commands are issued from the menubar or the Toolbar.

Menubar

You can issue the following commands from the menubar:

File: New, Open, Save, Save As, Print Setup, Print, Exit

Scripts: Run, Stop

Toolbar

The toolbar has buttons for running all of the commands listed above. To display this toolbar, select **View** > **Toolbars** > **Standard**.

New -- Clicking the New button brings up a new script template in the Work Area, so that a new script may be composed. If a modified script is open when the New button is clicked, Script Manager will ask if it should be saved.

Open -- Clicking the Open button brings up the Open dialog, so that a script can be loaded into the Work Area. If a modified script is open when the Open button is clicked, Script Manager will ask if it should be saved.

Save -- Clicking the Save button saves the script that is currently open in the Work Area.

Print -- Clicking the Print button causes the open script to print.

Cut -- Clicking Cut causes the selected text to be cut from the script.

Copy -- Clicking Copy causes the selected text to be copied to the clipboard.

Insert Clipboard Contents -- Pastes the contents of the clipboard into the script.

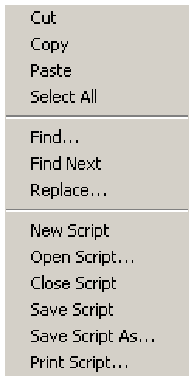
The Scripts toolbar has two buttons for running and stopping scripts:

Run -- Clicking the Run button saves (if needed) and executes the script that is currently displayed in the Work Area. While the script is running, the label on this button changes to Stop.

Stop -- Stops the currently running script.

Go to -- Clicking the Go To button opens the Go To dialog box. Here, users may enter a line number to go to a specific part of an open script. Line numbers are displayed on the bottom right of the Merlin's Wand application, on the status bar.

Script Manager Menu -- Right-clicking within the Work Area brings up the Script Manager menu. All filing and editing commands that can be performed in Script Manager can be accessed via this menu.



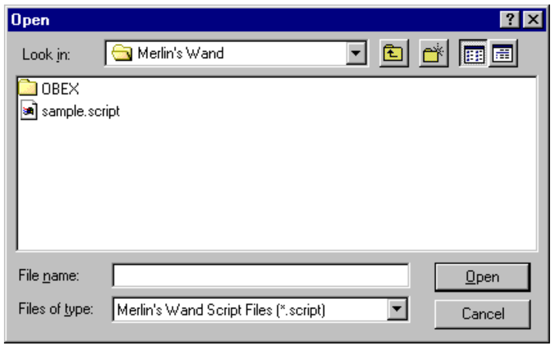
5.2 Running Scripts

Note: If Script Manager isn't enabled on your Merlin's Wand system, you will need to obtain a License Key from CATC before you can use it. See "Several frequently-used operations are bound to keyboard shortcuts." on page 15 for details.

Step 1 **Open Script Manager** by clicking the Script Manager icon on the toolbar or by selecting **Tools > Script Manager** from the menu bar.

Script Manager will open.

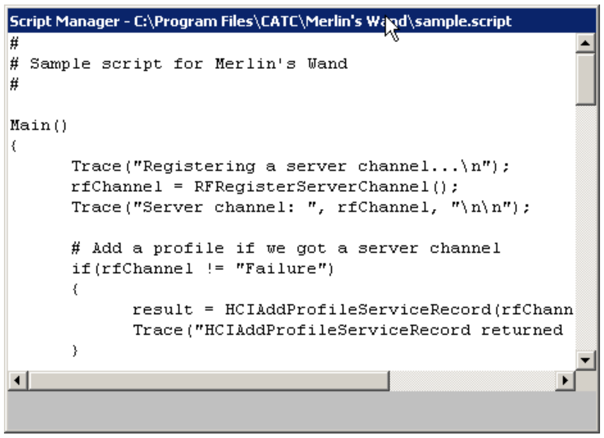
Step 2 **Open the script** by clicking the Open button in the Script Manager window or by selecting **File > Open Script…** from the menu bar.



The Open dialog will appear.

Step 3 Navigate to the desired file and click **Open**.

The script will display in Script Manager's Work Area.

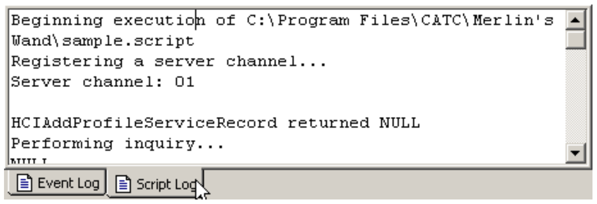

Script Manager - C:\Program Files\CATC\Merlin's Wand\sample.script

```
#
# Sample script for Merlin's Wand
#

Main()
{
        Trace("Registering a server channel...\n");
        rfChannel = RFRegisterServerChannel();
        Trace("Server channel: ", rfChannel, "\n\n");

        # Add a profile if we got a server channel
        if(rfChannel != "Failure")
        {
                result = HCIAddProfileServiceRecord(rfChann
                Trace("HCIAddProfileServiceRecord returned
        }
```

Step 4 Click the **Run** button.

Script execution will begin, and the label of the Run button will change to Stop. Pressing the Stop button terminates execution of the script.

The script's output can be viewed in the Script Log. If line numbers are referenced in the Script Log, double-clicking on the line number will move the cursor to that line in Script Manager.



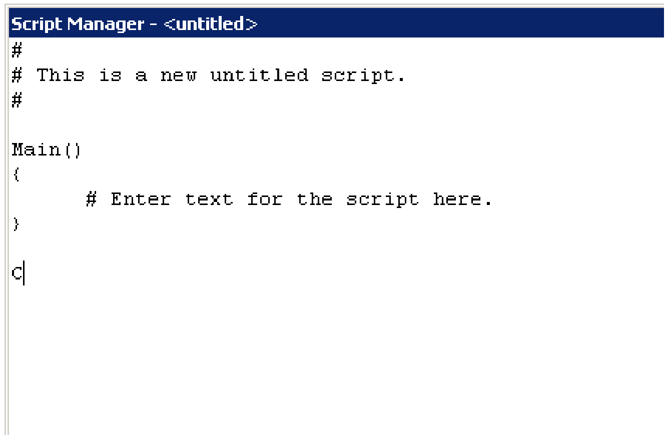
```
Beginning execution of C:\Program Files\CATC\Merlin's
Wand\sample.script
Registering a server channel...
Server channel: 01

HCIAddProfileServiceRecord returned NULL
Performing inquiry...
```

When the script has finished, the Stop button label will change back to Run.

5.3 Writing Scripts

Customized scripts can be written directly in Script Manager using Merlin's Wand Scripting Commands. This allows for automating sequences of commands. There are over 100 commands available for writing custom test sequences, including basic commands and commands for: pipes, HCI, L2CAP, SDP, RFCOMM, TCS, and OBEX. Detailed descriptions of the commands can be found in *Appendix C: Merlin's Wand Scripting Commands*, on page 155.

Step 1 **Open Script Manager** by clicking the Script Manager icon on the toolbar or by selecting **Tools > Script Manager** from the menu bar.



```
#
# This is a new untitled script.
#

Main()
{
	# Enter text for the script here.
}

c
```

By default, Script Manager opens an "untitled" script template in the Work Area for composing a new script. If Script Manager were already open, the Work Area could be cleared by pressing the **New** button in Script Manager or by selecting **File > New Script** from the menu bar.

Step 2 **Write** the script in Script Manager's Work Area.

Step 3 **Save** the script via the **Save Script As…** command on the

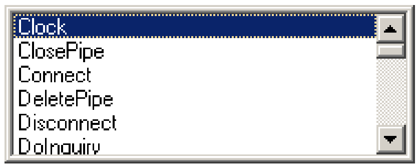
File menu or by clicking the **Save** button.

The Save As dialog will open. Enter a name for the script and save it as a Merlin's Wand Script file (*.script).

Step 4 If desired, **Close** the script by selecting **File > Close Script** from the menu bar.

5.4 Script Assistant

On a blank line, press control + spacebar on the keyboard to open a pop-up list of commands. Double click on a command to enter it into your script.



5.5 Sample Scripts

Sample scripts have been provided with Merlin's Wand to demonstrate how Script Manager works. The default location of the scripts is the directory where the application is installed, which is usually **C:\Program Files\CATC\Merlin's Wand**.

6. Device Search and Device List Pop-Up Menu

The Device Search and Device List Pop-Up Menu tools offer shortcut methods for steps that are commonly performed at the beginning of the connection process. They can be used for some commands that would otherwise need to be done in Command Generator.

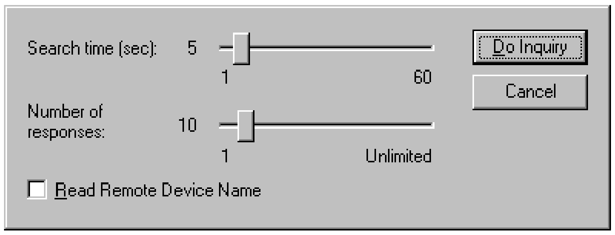
6.1 Device Search

Merlin's Wand can perform an inquiry to find local Bluetooth wireless technology devices via the Device Search tool. Information about the devices that are found are then shown in the Device List.

To perform a Device Search:

Step 1 Open the **Device Search dialog** by clicking the Device Search icon on the toolbar or by selecting **Tools > Device Search** from the menu bar.

The Device Search dialog will open.


Search time (sec): 5
Number of responses: 10
☐ Read Remote Device Name

Device Search presents the following search options:

Search Time -- Sets the duration of the inquiry, in seconds. The default search time is five seconds.

Number of Responses -- Sets the maximum number of responses for which data should be collected. The default number of responses is ten.

Read Remote Device Name -- Selecting this option will cause Merlin's Wand to collect name information from the remote devices it finds. This option is not selected by default.

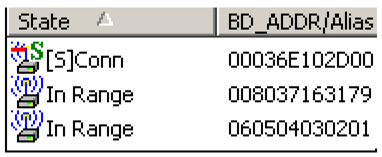
Note: The reading of names occurs after the search has finished; therefore, processing the entire search will take longer if this option is selected. For example, if the Search Time is set to 5 seconds, and 30 devices are found within 5 seconds, the entire search process will take much longer than 5 seconds because each device will be contacted individually and asked for its name. This could add considerable time to the search, especially if some of the devices found in the search have gone out of range or been turned off.

Step 2 (Optional) Set the values for Search Time, Number of Responses and Read Remote Device Name.

Step 3 Click **Do Inquiry**.

Merlin's Wand will search for devices.

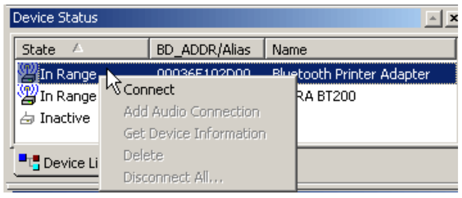
Step 4 To see the results of the search, click the **Device List** tab in the Device Status window. To see the commands and responses from the Inquiry, view the Event Log in the Logs window.



State	BD_ADDR/Alias
[S]Conn	00036E102D00
In Range	008037163179
In Range	060504030201

6.2 Device List Pop-Up Menu

The Device List Pop-Up Menu presents options for setting up ACL and audio (SCO) connections, displaying remote device information, and terminating connections. The Pop-Up Menu can be accessed by right-clicking on one or more devices in the Device List. It can be used as an alternative to Profile Wizard, Command Generator, and Script Manager for performing some commands.



Note!: The Device List Pop-Up Menu is not accessible while the Profile Wizard is running.

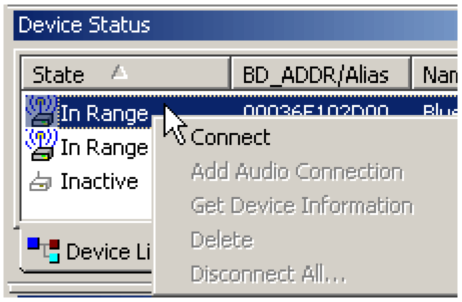
Create an ACL Connection

An HCI ACL connection to a remote device can be established via the Device List Pop-Up Menu.

Note: The following instructions assume that a Device Search has been performed and devices are displayed in the Device List. For information about performing a device search, please see Section 6.1, "Device Search," on page 61.

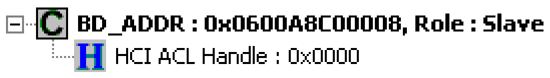
Step 1 **Open the Pop-Up Menu** by right-clicking on the target device in the Device List.

The Device List Pop-Up Menu will open.



Step 2 Choose **Connect** from the menu.

The status of the target device should change from **In Range** to **Connected** in the Device List. The Piconet tab should now indicate that Merlin's Wand has an ACL connection to the target device.


BD_ADDR : 0x0600A8C00008, Role : Slave
HCI ACL Handle : 0x0000

Establish an Audio Connection

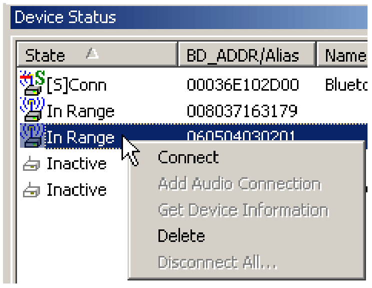
An HCI SCO connection to a device that supports audio connections can be established via the Device List Pop-Up menu.

Note: In order to verify that Merlin's Wand and a Bluetooth wireless audio device are successfully connected, a headset will need to be plugged into the audio port on Merlin's Wand. Be sure that the headset is plugged in before initializing the connection between Merlin's Wand and a Bluetooth device.

Note: The following instructions assume that a Device Search has been performed and devices are displayed in the Device List. For information about performing a device search, please see Section 6.1, "Device Search," on page 61.

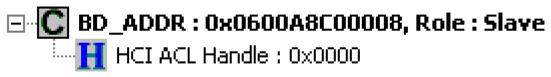
Step 1 **Open the Pop-Up Menu** by right-clicking on the target device in the Device List.

The Device List Pop-Up Menu will open.

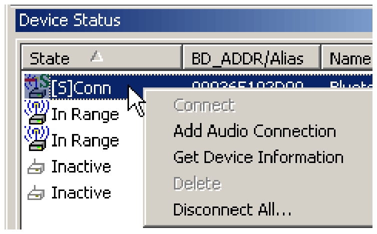


Step 2 Choose **Connect** from the menu.

The status of the target device should change from **In Range** to **Connected** in the Device List. The Piconet tab should now indicate that Merlin's Wand has an ACL connection to the target device.


BD_ADDR : 0x0600A8C00008, Role : Slave
HCI ACL Handle : 0x0000

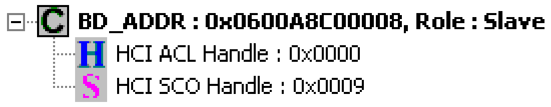
Step 3 **Reopen the Pop-Up Menu** by right-clicking on the target device in the Device List.



The Device List Pop-Up Menu will open. If the remote device supports audio connections and Merlin's Wand is connected to it, then the **Add Audio Connection** command should be available.

Step 4 Select Add Audio Connection from the menu.

The status of the target device will not change in the Device List; however, the Piconet tab should indicate that Merlin's Wand has an SCO connection to the device.

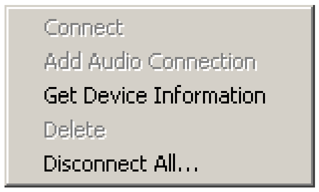

BD_ADDR : 0x0600A8C00008, Role : Slave
HCI ACL Handle : 0x0000
HCI SCO Handle : 0x0009

Step 5 To verify that Merlin's Wand and the Bluetooth device are successfully connected, speak into the microphone on one device and listen for audio on the other.

Display Device Information

Note: The following instructions assume that Merlin's Wand is currently connected to a remote device.

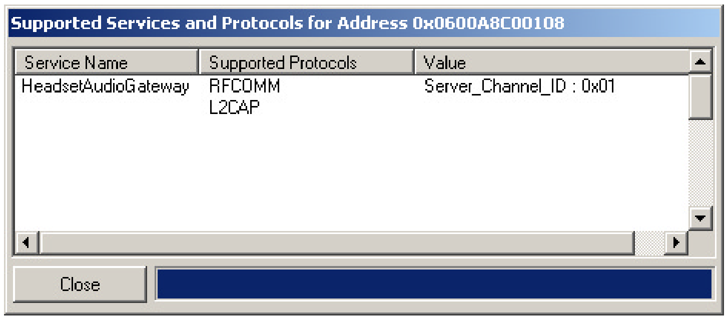
Step 1 Open the Pop-Up Menu by right-clicking on the target device in the Device List.



The Device List Pop-Up Menu will open.

Step 2 Select Get Device Information.

The Supported Services and Protocols window will open. The Service Name, Supported Protocols, and Value for the target device will be displayed in the window.


Supported Services and Protocols for Address 0x0600A8C00108

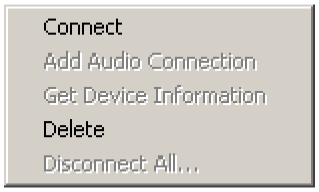
Service Name	Supported Protocols	Value
HeadsetAudioGateway	RFCOMM L2CAP	Server_Channel_ID : 0x01

Delete a Device

Devices that are not connected may be removed from the Device List via the Device List Pop-Up Menu. This is useful when there are many devices displayed in the Device List -- non-target devices can be deleted from the list, making it easier to navigate. This option can also be used to remove devices that are no longer in range, but are still displayed in the list.

Step 1 Open the Pop-Up Menu by right-clicking on one or more devices.

The Device List Pop-Up Menu will open.



Note: To delete more than one device at a time, either

(a) Select non-consecutive devices by Ctrl + clicking on each device to be deleted, or

(b) Select consecutive devices by Shift + clicking on the first and last devices to be deleted,

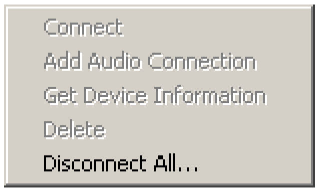
then right-click on one of the selected devices.

Step 2 Select Delete.

The device(s) will be removed from the Device List.

Disconnect All

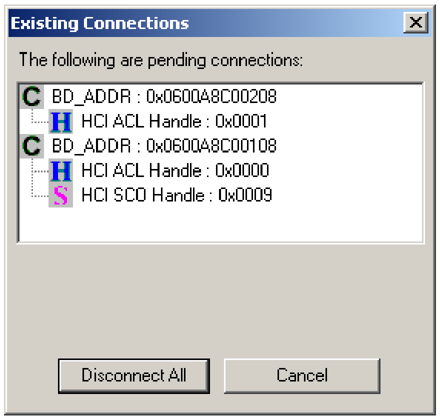
A fast and easy way to terminate all connections that Merlin's Wand has established with remote devices is to use the **Disconnect All…** command on the Device List Pop-Up Menu.



Step 1 **Open the Pop-Up Menu** by right-clicking on a device.

Step 2 Select **Disconnect All…**

The Existing Connections dialog will open, displaying all pending connections.



The following are pending connections:

```
C BD_ADDR : 0x0600A8C00208
  H HCI ACL Handle : 0x0001
C BD_ADDR : 0x0600A8C00108
  H HCI ACL Handle : 0x0000
  S HCI SCO Handle : 0x0009
```

Step 3 Click the **Disconnect All** button in the Existing Connections dialog to close the connections, or click **Cancel** to leave them open.

Note: When switching between Profile Wizard, Command Generator and Script Manager, all connections that have been established between Merlin's Wand and another Bluetooth device should be closed. However, expert users may choose to leave the connections open. If a connection is left open and you attempt to switch tools, Merlin's Wand will prompt you to close the connections. Choosing **Disconnect All** will close the connections. Choosing **Cancel** will leave the connections open, but some commands might not work properly in the other tool. When switching to Profile Wizard, any open connections *must* be closed.

7. Data Transfer Manager and Data Pipes

Data Transfer Manager is a special tool for creating *pipes*. A pipe is a file or message that has been specially prepared for transmission over an RFCOMM or L2CAP channel. Pipes are necessary with these protocols because RFCOMM and L2CAP only transfer raw data. Pipes are set up to designate the source of the raw data -- either a file or text entered by the user.

Data Transfer Manager is designed to work in conjunction with Command Generator and Script Manager. Use Data Transfer Manager to prepare data, then use Command Generator or Script Manager to transfer it.

Data Transfer Manager is also used to view any data that is being received by Merlin's Wand over L2CAP or RFCOMM channels. If data is being received, the Data Transfer Manager window will automatically create a Receive Pipe and display the arriving data.

Data Transfer Manager contains the following elements:

Data Transmit page -- The Data Transmit page is used to create pipes. Pipes can be created from files or text. For more information on data transmit pipes, see Section 7.1, "Creating Data Pipes," on page 67 and Section 7.2, "Using Data Pipes," on page 68.

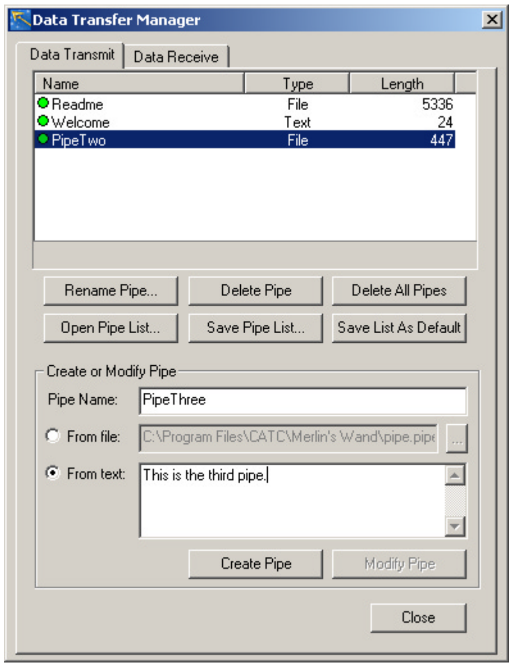
Data Receive page -- The Data Receive page is used to view data that has been received by Merlin's Wand. For more information, see Section 7.3, "Receive Pipes," on page 70.

7.1 Creating Data Pipes

Data Transfer Manager is used to prepare data for transmission over RFCOMM and L2CAP channels. To prepare the data, pipe files are created.

Step 1 **Open Data Transfer Manager** by clicking on the Data Transfer Manager icon on the Toolbar or selecting **Tools > Data Transfer Manager** from the menu bar.

Data Transfer Manager will open, displaying the Data Transmit page.



Name	Type	Length
Readme	File	5336
Welcome	Text	24
PipeTwo	File	447

Pipe Name: PipeThree
From file: C:\Program Files\CATC\Merlin's Wand\pipe.pipe
[✓ From text:] This is the third pipe.

Step 2 **Name the pipe** by typing a name into the text box labeled **Pipe Name**.

Step 3 *To create a pipe from a file:*

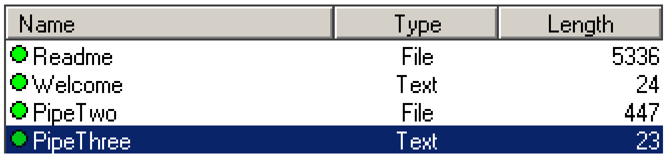
Select the **From file** radio button. Type in a filename and path or navigate to the desired file by clicking the browse button ... to bring up the Open dialog.

To create a pipe from text:

Select the **From text** radio button and type text into the box to its right.

Step 4 Click the **Create Pipe** button.

The pipe will be created, and its name, type and length will be displayed in the list at the top of the Data Transmit page.



Name	Type	Length
Readme	File	5336
Welcome	Text	24
PipeTwo	File	447
PipeThree	Text	23

7.2 Using Data Pipes

Data Transfer Manager works in conjunction with Command Generator and Script Manager to provide an easy way to transfer files between Merlin's Wand and a Bluetooth wireless device over an RFCOMM or L2CAP channel.

Transfer Data Using Command Generator

Note: A data pipe needs to be created in Data Transfer Manager before the data can be transferred. See Section 7.1, "Creating Data Pipes," on page 67 to learn how to do this.

Step 1 **Open Command Generator** by clicking on the Command Generator icon on the Toolbar or selecting **Tools > Command Generator** from the menu bar.

Step 2 **Establish an RFCOMM or L2CAP connection** with the target device.

To establish an RFCOMM connection:

(a) Using the Device List pop-up menu, create an ACL connection. To find out how to do this, see "Create an ACL Connection" on page 62.

(b) In Command Generator, open the RFCOMM tab.

(c) Select OpenClientChannel from the list of commands. Enter the HCI_Handle (this is shown on the Piconet tab) and ServerChannel parameters. The default values for MaxFrameSize and Credit can be used. When the parameter values are all entered, press Execute.

The RFCOMM connection will show up in the Piconet.

To establish an L2CAP connection:

(a) Using the Device List pop-up menu, create an ACL connection. To find out how to do this, see "Create an ACL Connection" on page 62.

(b) In Command Generator, open the L2CAP tab.

(c) Select RegisterPsm from the list of commands. Enter the PSM and Receive MTU parameters, then press Execute. Note: RegisterPsm must also be executed for the target device.

(d) Select ConnectRequest from the list of commands. Enter the HCI_Handle, PSM, and Receive MTU parameters, then press Execute.

The L2CAP connection will display on the Piconet tab.

Step 3 **Select SendData** from the RFCOMM or L2CAP menu, depending on which type of connection was established in Step 2.

The first two Parameters Combo Boxes will become activated, indicating that the (HCI / DLCI) for RFCOMM, or CID for L2CAP, and Data Pipe parameters are required for this command.

Step 4 **Enter or select** the appropriate **(HCI / DLCI) or CID channel** and **Data Pipe name** in the Parameters Combo

Boxes.

Step 5 Click the **Execute** button to send the data pipe.

The Event Log will show the transfer of data from Merlin's Wand to the target device.

Access Pipes Using Script Manager

There are functions available in the scripting API to access pipes. They are: OpenPipe, ClosePipe, ReadPipe, WritePipe, and DeletePipe. Please see Section C.3, "Pipe Commands," on page 160 for more information.

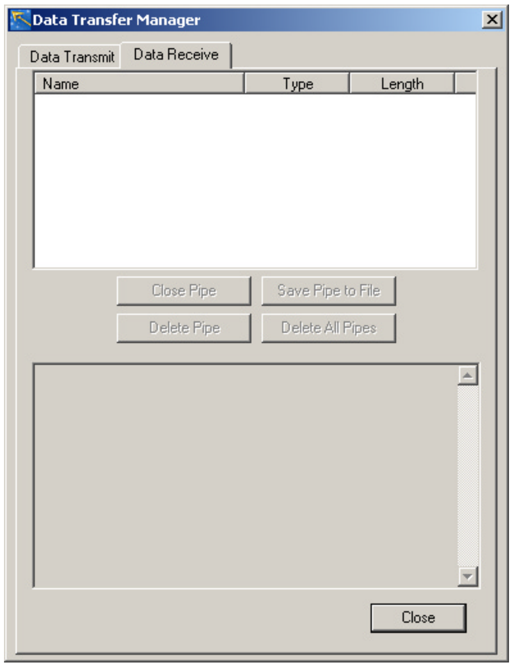
For a demonstration of using RFCOMM to wait for and receive a data pipe, see **Sample-2.script** in Script Manager.

Note: A data pipe needs to be created in Data Transfer Manager before the data can be transferred. See Section 7.1, "Creating Data Pipes," on page 67 to learn how to do this.

7.3 Receive Pipes

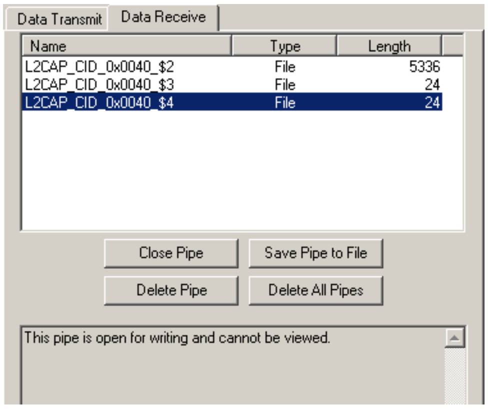
Receive pipes are created automatically in the Data Receive page of Data Transfer Manager when Merlin's Wand receives data from an L2CAP or RFCOMM connection.

A receive pipe is a pipe that is used to receive data until the connection terminates. At that point, the pipe can be closed, saved to a file, or deleted. Note that unless a receive pipe is closed, any additional data that's received will be put into that pipe, even if the new data represents a different file transfer. To ensure that different files will be put into separate pipes, each pipe should be closed after a connection has completed. This way, a new receive pipe will be created when subsequent data arrives.



7.4 Closing Pipes

Receive pipes on the Data Receive page of Data Transfer Manager can be closed. Closing a receive pipe prevents additional data from being placed in it. Closing it also allows its contents to be viewed in the bottom window of the Data Receive page. For any pipe that isn't closed, this message will appear in the window: "This pipe is open for writing and cannot be viewed."



Name	Type	Length
L2CAP_CID_0x0040_$2	File	5336
L2CAP_CID_0x0040_$3	File	24
L2CAP_CID_0x0040_$4	File	24

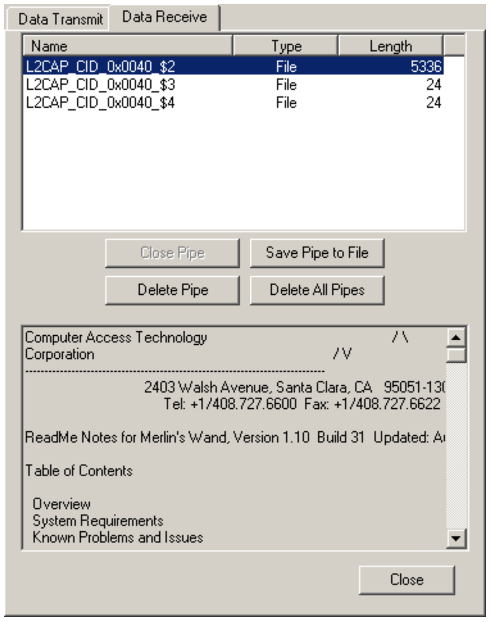
This pipe is open for writing and cannot be viewed.

To close a receive pipe:

Step 1 **Select the pipe** that is to be closed.

Step 2 Click **Close Pipe**.

The pipe will be closed, and its contents will be shown in the bottom window.



Name	Type	Length
L2CAP_CID_0x0040_$2	File	5336
L2CAP_CID_0x0040_$3	File	24
L2CAP_CID_0x0040_$4	File	24

```
Computer Access Technology                    /\
Corporation                                  /V
-------------------------------------------------
                2403 Walsh Avenue, Santa Clara, CA   95051-13
                   Tel: +1/408.727.6600  Fax: +1/408.727.6622

ReadMe Notes for Merlin's Wand, Version 1.10  Build 31  Updated: A

Table of Contents

 Overview
 System Requirements
 Known Problems and Issues
```

7.5 Saving Data Pipes

Data Transfer Manager can save data pipes that are prepared for transmission as well as data pipes that are received.

Saving Data Transmit Pipe Lists

Step 1 (Optional) **Delete** all pipes. If pipes are already displayed on the Data Transmit page of Data Transfer Manager, any newly created pipes will be added to the displayed list. To create an entirely new list of pipes, the currently displayed list should be deleted.

Note: The default pipe list is automatically loaded into the Data Transmit page when the Merlin's Wand application is opened. If no list has been saved as default, then no list will be loaded.

If the application hasn't been shut down since the last time that Data Transfer Manager was used, then the last list that was open in the Data Transmit page will be displayed the next time the tool is opened.

Step 2 **Create** one or more data pipes.

Step 3 Click **Save Pipe List...** to bring up the Save As dialog. Enter a file name and save the list as a Merlin's Wand Pipe File [*.pipe].

Saving a Default List

Step 1 **Create** one or more data pipes **or open** a pipe list.

Step 2 Click **Save List As Default**.

The list will be saved as `default.pipe`. That list is loaded into the Data Transmit page when the Merlin's Wand application is opened. However, if the user exits Data Transfer Manager but doesn't exit the application, the last list that was open in the Data Transmit page will be the one displayed when the tool is next accessed.

Saving Data Receive Pipes

Step 1 **Select a pipe** in the Data Receive page of Data Transfer manager.

Step 2 Click **Save Pipe to File**.

The Save As dialog will come up. Enter a file name, including the file type extension, then click Save.

7.6 Deleting Pipes

Note: Deleting pipes removes them from the list displayed in Data Transfer Manager. If the pipes were previously saved in a pipe list file, deleting them in Data Transfer Manager won't delete them from the file. To delete pipes from a pipe list file, first delete the pipes, then save the pipe list.

To delete a pipe:

Step 1 **Select the pipe** to be deleted.

Step 2 Click **Delete Pipe**.

The pipe will be deleted from the displayed pipe list.

To delete all pipes:

Step 1 Click **Delete All Pipes**.

A warning dialog will come up, asking, "Are you sure you want to delete all pipes from the list?" Click **Yes** to delete the pipes.

All pipes will be cleared from the display.

7.7 Opening Pipe Lists

To open a pipe list in the Data Transmit page of Data Transfer Manager:

Step 1 Click **Open Pipe List...**

Note: If pipes are already displayed, a dialog box will ask, "Delete current pipes before adding new pipes from pipe list file?" Choose Yes to delete the displayed pipes, or click No to leave those pipes displayed.

The Open dialog will come up.

Step 2 **Select a file**, then click **Open**.

The selected list will be displayed on the Data Transmit page.

7.8 Renaming Pipes

To rename a pipe displayed in the Data Transmit page of Data Transfer Manager:

Step 1 **Select the pipe** to be renamed.

Step 2 Click **Rename Pipe...**

The Rename Pipe dialog will appear.

Step 3 **Enter a new name** for the pipe, then click **OK**.

Note: Renaming a pipe changes its name in Data Transfer Manager. To change a pipe's name in a pipe list file, first rename the pipe, then save the pipe list.

7.9 Modifying Pipes

Existing pipes may be modified on the Data Transmit page of Data Transfer Manager. A pipe created from a file can be modified either by associating it with a different file or by changing it to a text-based pipe. A pipe created from text can be modified either by editing the text or by changing the pipe to a file-based pipe.

Note: Modifying pipes changes them in Data Transfer Manager. However, if the pipes were previously saved in a pipe list file, modifying them in Data Transfer Manager won't change them in the saved file. To modify pipes in a pipe list file, first modify the pipes, then save the pipe list.

Step 1 **Select the pipe** that is to be modified.

Step 2 ***To associate a pipe with a different file or change a text-based pipe to a file-based pipe:***

- With the "From file" radio button selected, enter a new filename and path or navigate to the new file by clicking the browse button ... to bring up the Open dialog. **Select a new file** and click Open.

To modify text or change a file-based pipe to a text-based pipe:

- With the "From text" radio button selected, **change, add or delete text** in the text entry box.

Step 3 Click **Modify Pipe** to implement the changes.

8. Using Merlin to Record Merlin's Wand Traffic

It's possible to control the CATC Merlin Bluetooth Protocol Analyzer via Merlin's Wand. The two can be used together to capture real-time test sequence results, as is required by the Bluetooth SIG to provide evidence of product compliance to the specification.

Merlin's Wand has built-in functionality for controlling the Merlin protocol analyzer. Through Merlin's Wand, a Bluetooth recording session can be set up on Merlin, even if the Merlin application runs on a remote computer.

8.1 Set Up a Remote Machine

If Merlin's Wand will be used to run Merlin on a remote machine, DCOM and accessibility properties on the remote machine must be properly configured. The configuration procedures differ slightly for different operating systems.

Note: If Merlin's Wand will be used only to run Merlin on the same computer that is running Merlin's Wand, skip to "Connect to Merlin with Merlin's Wand" on page 79.

Windows 98/Me Operating Systems

Use this procedure to configure DCOM properties to run a Merlin analyzer remotely on a machine running Windows 98 or Windows Me. All the steps should be performed on the remote machine.

Step 1 **Open the Merlin application** on the remote machine in order to register it with COM, and then close the application.

Step 2 **Download and install** dcom98.exe and dcm95cfg.exe DCOM configuration utilities from Microsoft. They are available via the following links:

http://download.microsoft.com/msdownload/dcom/98/x86/en/dcom98.exe

http://download.microsoft.com/msdownload/dcom/98/x86/en/dcm95cfg.exe

Step 3 Select **Start > Run** from the Windows taskbar.

The Run dialog will open.

Step 4 **Enter "dcomcnfg"** in the Open combo box and **press OK**.

The Distributed COM Configuration Properties dialog will open.

Step 5 On the Applications tab, **select Merlin** from the list of applications.

Step 6 Select the **Default Properties tab** and make sure that "Enable Distributed COM on this computer" is checked.

Step 7 Select the **Default Security tab** and make sure that "Enable remote connection" is checked.

Step 8 **Click OK**.

Windows NT®/2000 Operating Systems

Use this procedure to configure DCOM and accessibility properties to run a Merlin analyzer remotely on a machine running Windows NT 4.0 or Windows 2000. All the steps should be performed on the remote machine.

Step 1 **Open the Merlin application** on the remote machine in order to register it with COM, and then close the application.

Step 2 Select **Start > Run** from the Windows taskbar.

The Run dialog will open.

Step 3 **Enter "dcomcnfg"** in the Open combo box and **press OK**.

The Distributed COM Configuration Properties dialog will open.

Step 4 On the Applications tab, **select Merlin** from the list of applications.

Step 5 Click the **Properties** button.

The Merlin Properties dialog will open.

Step 6 Go to the **Security tab**.

Step 7 Select the "**Use custom launch permissions**" radio button and click the associated **Edit...** button.

The Registry Value Permissions dialog will open.

Step 8 Make sure **Allow Launch** is selected in the Type of Access drop-down list.

Step 9 Click **Add...**

The Add Users and Groups dialog will open.

Step 10 **Select names** from the Names list and click **Add** to include them in the Add Names list.

Step 11 When all desired user names have been added, **click OK**.

Step 12 **Click OK** on each of the three dialogs that are still open.

Step 13 _For Windows NT_:

(a) Select **Start > Settings > Control Panel** on the Windows taskbar.
The Control Panel window will open.

(b) Double-click on **Services**.
The Services dialog will open.

(c) Select **Remote Procedure Call (RPC) Locator** and select **Action > Properties** from the menu bar.
The Remote Procedure Call (RPC) Locator Properties dialog will open.

(d) Click **Start** on the General tab, then click **OK**.
The status for Remote Procedure Call (RPC) Locator should now be set to "Started" in the Services dialog.

(e) Select R**emote Procedure Call (RPC) Service** and select **Action > Properties** from the menu bar.
The Remote Procedure Call (RPC) Service Properties dialog will open.

(f) Click **Start** on the General tab, then click **OK**.
The status for Remote Procedure Call (RPC) Service should now be set to "Started" in the Services dialog.

For Windows 2000:

(a) Select **Start > Settings > Control Panel** on the Windows taskbar.
The Control Panel window will open.

(b) Double-click on **Administrative Tools**.
The Administrative Tools window will open.

(c) Double-click on **Services**.
The Services dialog will open.

(d) Select **Remote Procedure Call (RPC)** and select **Action > Properties** from the menu bar.
The Remote Procedure Call (RPC) Properties dialog will open.

(e) Click **Start** on the General tab, then click **OK**.
The status for Remote Procedure Call (RPC) should now be set to "Started" in the Services dialog.

(f) Select **Remote Procedure Call (RPC) Locator** and select **Action > Properties** from the menu bar.
The Remote Procedure Call (RPC) Locator Properties dialog will open.

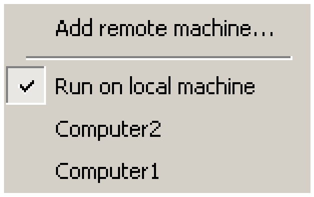
(g) Click **Start** on the General tab, then click **OK**.
The status for Remote Procedure Call (RPC) Locator should now be set to "Started" in the Services dialog.

8.2 Set Up Connection Options

In addition to establishing connections, the Connect/Disconnect Merlin Bluetooth Analyzer button provides several options for configuring the connection between Merlin's Wand and Merlin. To see the options, click on the options arrow on the right side of the button.

Run Merlin on a Local Machine

To run Merlin on a local machine, click the options arrow on the right side of the Connect/Disconnect Merlin button and make sure that "Run on local" is selected in the Connect/Disconnect options menu. When selected, a checkmark appears next to it.

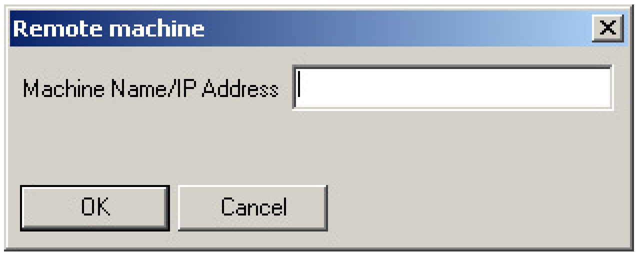


Add a Remote Machine

Merlin's Wand can be configured to control a Merlin analyzer that is running on a remote computer. Before Merlin's Wand can connect to Merlin, the remote machine that runs Merlin must be added to the Connect/Disconnect options menu in Merlin's Wand.

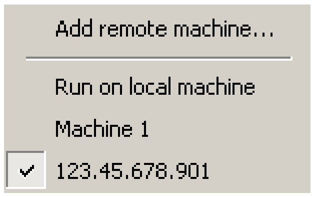
Step 1 Click the options arrow on the right side of the Connect/Disconnect Merlin button and select "**Add remote machine...**" from the options menu.

The Remote Machine dialog will open.



Step 2 Enter the **Internet machine name or IP address** for the machine on which Merlin is running and click OK.

Note: If the machine name is used and Merlin's Wand is subsequently unable to connect to Merlin, then the IP address must be used instead.



The machine name/IP address will now be listed on the Connect/Disconnect options menu. By default, it will be selected, as indicated by the checkmark that appears to the left of the name/IP address.

8.3 Start Merlin

This step is required only when running Merlin on a remote machine that uses Windows 98 or Windows Me. In such cases, Merlin's Wand cannot start or stop the Merlin application, although it can control Merlin once it is running. **Be sure to start Merlin before connecting to it on a machine running Windows 98 or Windows Me.**

8.4 Connect to Merlin with Merlin's Wand

Note: Before Merlin's Wand can connect to Merlin, the connection options, DCOM, and accessibility properties may need to be configured. Please refer to "Set Up Connection Options" on page 78 and "Set Up a Remote Machine" on page 75 for more information.

Step 1 To connect to Merlin Bluetooth Protocol Analyzer, click on the **Connect/Disconnect** button.

Merlin's Wand will connect to Merlin. The status bar at the bottom of the Merlin's Wand application will indicate that Merlin is connected, along with the Merlin software version. The Connect/Disconnect button will remain "pressed down" while Merlin's Wand and Merlin are connected.

8.5 Set Merlin Recording Options

The recording options file [*.rec] that Merlin should use can be specified through Merlin's Wand. If a recording options file isn't specified through Merlin's Wand, Merlin will use either its default .rec file or the options file that was last loaded into the current instance of Merlin.

Note: The .rec file has to be configured and saved in Merlin before it can be specified through Merlin's Wand.

Step 1 **Share the folder** that contains the file.

(a) In Windows Explorer or My Computer, navigate to the folder that contains the options file.

(b) Right-click on the folder and select Properties, or select File > Properties from the menu bar.

The Properties dialog will open.

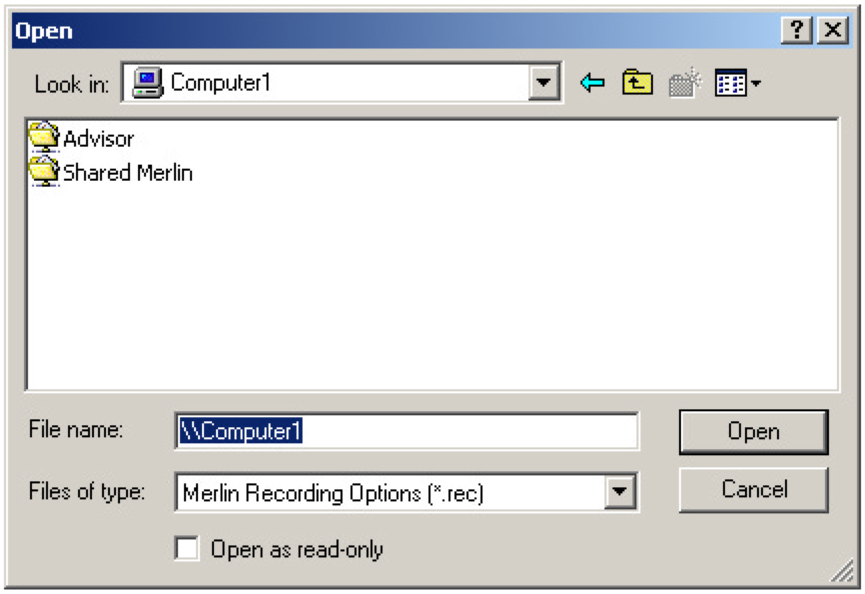
(c) Go to the Sharing tab in the Properties dialog.

(d) Enable the "Share this folder" option and make sure that the folder is accessible by both the machine running Merlin and the machine running Merlin's Wand.

(e) Click OK.

Step 2 Click the **Set Recording Options** button.

The Open dialog will be displayed.



File name: \\Computer1

Files of type: Merlin Recording Options (*.rec)

Step 3 Use the **Look in** field at the top of the dialog box to browse to the desired file via Network Neighborhood
-or-

In the **File name** field, type \\ followed by the name of the computer on which the file is located (for example, \\Computer1). Press Enter to display all shared folders, then navigate to the desired file.

Note: A full network path must be used in order to specify the options file through Merlin's Wand, whether the file is local to the machine running Merlin or located on a different computer.

Step 4 Click **Open**.

The path and filename of the recording options file will now be listed on the Set Recording Options drop-down menu. By default, that file will be selected, as indicated by the checkmark that appears to the left of the path and filename.

8.6 Set Merlin Display Options

The display options file [*.opt] that Merlin should use can be specified through Merlin's Wand. If a display options file isn't specified through Merlin's Wand, Merlin will use either its default .opt file or the options file that was last loaded into the current instance of Merlin.

Note: The .opt file has to be configured and saved in Merlin before it can be specified through Merlin's Wand.

Step 1 **Share the folder** that contains the file.

(a) In Windows Explorer or My Computer, navigate to the folder that contains the options file.

(b) Right-click on the folder and select Properties, or select File > Properties from the menu bar.

The Properties dialog will open.

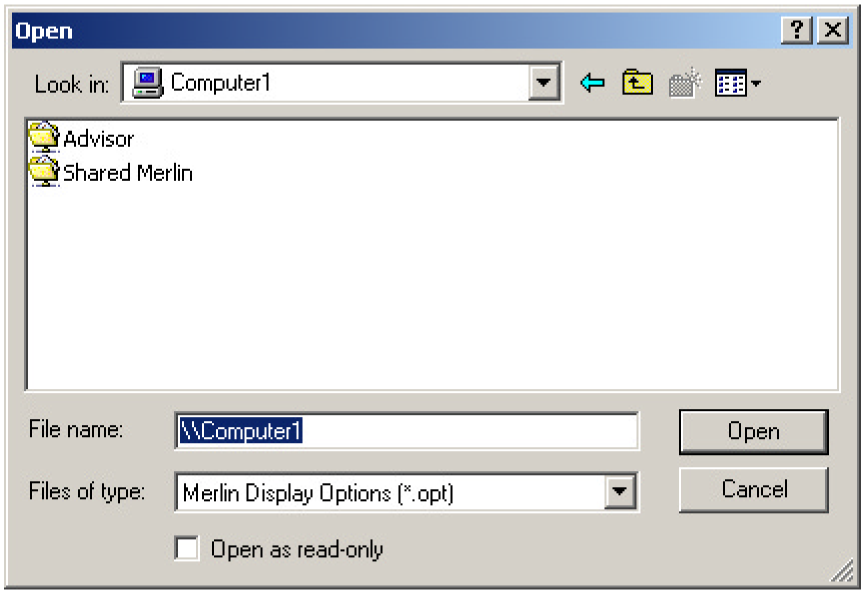
(c) Go to the Sharing tab in the Properties dialog.

(d) Enable the "Share this folder" option and make sure that the folder is accessible by both the machine running Merlin and the machine running Merlin's Wand.

(e) Click OK.

Step 2 Click the **Set Display Options** button.

The Open dialog will be displayed.



Computer1
Advisor
Shared Merlin
File name: \\Computer1
Files of type: Merlin Display Options (*.opt)
Open as read-only

Step 3 Use the **Look in** field at the top of the dialog box to browse

to the desired file via Network Neighborhood
-or-

In the **File name** field, type \\ followed by the name of the computer on which the file is located (for example, \\Computer1). Press Enter to display all shared folders, then navigate to the desired file.

Note: A full network path must be used in order to specify the options file through Merlin's Wand, whether the file is local to the machine running Merlin or located on a different computer.

Step 4 Click **Open**.

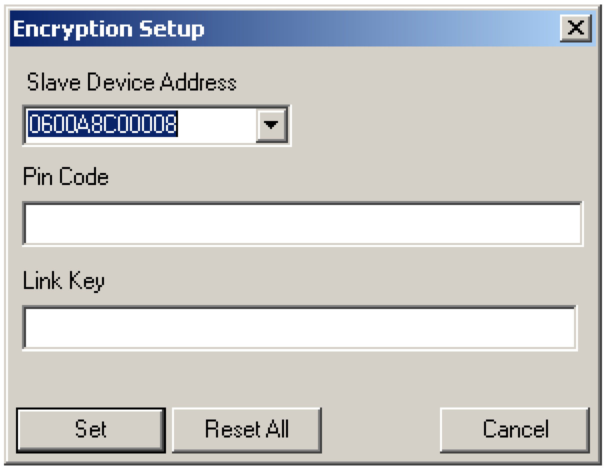
The path and filename of the display options file will now be listed on the Set Display Options drop-down menu. By default, that file will be selected, as indicated by the checkmark that appears to the left of the path and filename.

8.7 Set Merlin Encryption Options

Merlin's Wand can set up Merlin to decode encrypted transmissions.

Step 1 Open the Encryption Setup dialog by pressing the **Set Merlin encryption options** button.

The Encryption Setup dialog will open.



Step 2 **Select the Slave Device Address** from the drop-down list, or enter it into the combo box.

Step 3 **Enter the PIN Number** for the slave device in the PIN Code text box.

or

Enter the Link Key for the master-slave connection in the Link Key text box.

Step 4 Press the **Set** button to apply the encryption setup.

8.8 Start a Merlin Recording Session

To begin a Merlin Bluetooth Analyzer recording session, press the Record REC button on the Merlin toolbar.

8.9 Stop a Merlin Recording Session

To stop a Merlin Bluetooth Analyzer recording session, press the Stop STOP button on the Merlin toolbar.

8.10 Disconnect from Merlin Bluetooth Protocol Analyzer

To disconnect from Merlin Bluetooth Protocol Analyzer, click on the Connect/Disconnect button.

Merlin's Wand will disconnect from Merlin.

8.11 Troubleshooting

"Server Busy" When Attempting to Launch Merlin

"Server Busy" message appears when attempting to launch Merlin on a remote Windows 98 or Windows Me system.

- Make sure Merlin is running on the remote machine before clicking the Connect/Disconnect Merlin button in Merlin's Wand. Merlin's Wand cannot start or stop Merlin on a remote Windows 98 or Windows Me system; it can only control Merlin once Merlin is running on the remote machine.
- Make sure DCOM is properly installed and configured on the remote machine, as described in the section DCOM Configuration for Windows 98/Me Systems.
- If the message box won't go away, close Merlin's Wand (press Ctrl+Alt+Delete and close the program directly or through the Task Manager). Then, restart Merlin's Wand and be sure to start Merlin on the remote machine before retrying the operation.

"Server Busy" message appears when attempting to launch Merlin on a remote Windows NT or Windows 2000 system.

- This message may appear the first time Merlin is launched on the remote machine. It can be safely ignored. Merlin will start normally on the remote machine. Clicking the "Switch To" button in the message box will cause the message to disappear and the Start menu to appear. Return to Merlin's Wand and proceed normally.

"Server Execution Failed" When Attempting to Launch Merlin

"Server execution failed" message appears when attempting to launch Merlin on a remote Windows 98 or Windows Me system.

- Make sure Merlin is running on the remote machine before clicking the Connect/Disconnect Merlin button in Merlin's Wand.

"The Object Exporter Specified Was Not Found" When Attempting to Launch Merlin

"The object exporter specified was not found" message appears when attempting to launch Merlin from a Windows 2000 system.

- If Merlin is already running on the remote machine, try to close it. If a message appears indicating that an automation client is connected to the application, close Merlin on the remote machine, close Merlin's Wand on the local machine, and try again. If the problem persists, restart the remote machine.
- Make sure the local computer (the Windows 2000 system running Merlin's Wand) can reach the remote machine by using its full computer name, such as <computername.domain>. Open a command prompt and use the PING command to determine this. If the local computer cannot communicate with the remote machine, consult a system administrator for assistance.
- If the remote machine is running TCP/IP, make sure it has an assigned IP address. Consult a system administrator for assistance in determining this and, if necessary, assigning an IP address.

9. Contact and Warranty Information

9.1 Contact Information

Mailing address

Computer Access Technology Corporation
Customer Support
2403 Walsh Avenue
Santa Clara, CA 95051-1302
USA

Online support

http://www.catc.com/

E-mail address

support@catc.com

Telephone support

+1/800.909.2282 (USA and Canada)
+1/408.727.6600 (worldwide)

Fax

+1/408.727.6622 (worldwide)

Sales information

sales@catc.com

9.2 Warranty and License

Computer Access Technology Corporation (hereafter CATC) warrants this product to be free from defects in material, content, and workmanship, and agrees to repair or replace any part of the enclosed unit that proves defective under these terms and conditions. Parts and labor are warranted for one year from the date of first purchase.

The CATC software is licensed for use on a single personal computer. The software may be copied for backup purposes only.

This warranty covers all defects in material or workmanship. It does not cover accidents, misuse, neglect, unauthorized product modification, or acts of nature. Except as expressly provided above, CATC makes no warranties or conditions, express, implied, or statutory, including without limitation the implied warranties of merchantability and fitness for a particular purpose.

CATC shall not be liable for damage to other property caused by any defects in this product, damages based upon inconvenience, loss of use of the product, loss of time or data, commercial loss, or any other damages, whether special, incidental, consequential, or otherwise, whether under theory of contract, tort (including negligence), indemnity, product liability, or otherwise. In no event shall CATC's liability exceed the total amount paid to CATC for this product.

CATC reserves the right to revise these specifications without notice or penalty.

Appendix A: Command Generator Command Descriptions

A.1 HCI Command Descriptions

Note **"N/A"** means Not Applicable. This indicates that the specified command does not have a parameter.

HCI Link Control Commands

Accept_Connection_Request

Used to accept a new incoming connection request. Execute this command before connection request from another device. By default, all connection requests are accepted.

Command Parameters	Examples	Comments
N/A		

Return Events
Accept_Connection_Request_Complete

Add_SCO_Connection

Will cause the link manager to create an SCO connection in addition to the existing ACL connection.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Packet Type	0x0080	Possible packet types:
		HV1=0x0020
		HV2=0x0040
		HV3=0x0080

Return Events
Add_SCO_Connection_Complete
Add_SCO_Connection_Error

Authentication_Requested

Used to initiate authentication between the two devices associated with the specified HCI_Handle.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Authentication_Error
Authentication_Complete

Change_Connection_Link_Key

Used to force both connected devices to generate a new link key.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	Range: 0x0000-0x0EFF

Return Events
Change_Connection_Link_Key_Error
Change_Connection_Link_Key_Complete

Change_Connection_Packet_Type

Used to change which packet types can be used for a connection that is currently established.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Packet_Type	0x0008	Possible Values: ACL 0x0008 = DM1 0x0010 = DH1 0x0400 = DM3 0x0800 = DH3 0x4000 = DM5 0x8000 = DH5 SCO HV1: 0x0020 HV2: 0x0040 HV3: 0x0080 or any combination

Return Events
Change_Connection_Packet_Type_Error
Change_Connection_Packet_Type_Complete

Create_Connection

Create_Connection will cause the link manager to create an ACL connection to the Bluetooth wireless device with the BD_ADDR specified by the command parameters.

Command Parameters	Examples	Comments
BD_ADDR	010203040506	Enter in HEX as shown.
Allow_Role_Switch		

Return Events
Create_Connection_Complete
Create_Connection_Error

Disconnect

Disconnect is used to terminate an existing connection.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Disconnection_Complete
Disconnection_Failed

Exit_Periodic_Inquiry_Mode

Exit_Periodic_Inquiry_Mode is used to end the Periodic Inquiry mode when Merlin's Wand is in Periodic Inquiry mode.

Command Parameters	Examples	Comments
N/A		

Return Events
Exit_Periodic_Inquiry_Mode_Complete
Exit_Periodic_Inquiry_Mode_Error

Inquiry

Inquiry will cause Merlin's Wand to enter Inquiry mode and discover other nearby Bluetooth devices.

Command Parameters	Examples	Comments
Inquiry_Length	8	
Num_Responses	10	

Return Events
Inquiry_Complete
Inquiry_Result
Inquiry_Error

Inquiry_Cancel

Inquiry_Cancel will cause Merlin's Wand to stop the current Inquiry if the Bluetooth device is in Inquiry mode.

Command Parameters	Examples	Comments
N/A		

Return Events
Inquiry_Canceled
Inquiry_Error

Periodic_Inquiry_Mode

Periodic_Inquiry_Mode is used to configure Merlin's Wand to perform a periodic Inquiry based on a specified period range.

Note: Max_Period_Length > Min_Period_Length > Inquiry Length.

Command Parameters	Examples	Comments
Max Period Length	10	Range: 0x03 – 0xFFFF
Min Period Length	9	Range: 0x02 – 0xFFFE
Inquiry Length	8	Range: 0x01 – 0x30
Num of Responses	10	Range: 0-255 (0=Unlimited number of responses)

Return Events
Periodic_Inquiry_Mode_Complete
Periodic_Inquiry_Mode_Error

PIN_Code_Request_Negative_Reply

PIN_Code_Request_Negative_Reply is used to reply to a PIN Code Request event from the Host Controller when the Host cannot specify a PIN code to use for a connection. This command should be executed before PIN_Code_Request event is received. By default, the PIN_Code_Request event is rejected.

Command Parameters	Examples	Comments
BD_ADDR	0x010203040506	

Return Events
Command_Complete

PIN_Code_Request_Reply

PIN_Code_Request_Reply is used to reply to a PIN Code Request event from the Host Controller and specifies the PIN code to use for a connection. This command should be executed before Pin_Code Request event is received. By default, the Pin_Code Request event is rejected.

Command Parameters	Examples	Comments
PIN_Code	1234	PIN_Code is a string character that can be up to 128 bits in length
BD_ADDR	0x010203040506	

Return Events
Command_Complete

Read_Clock_Offset

Read_Clock_Offset allows the Host to read the clock offset of remote devices.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Read_Clock_Offset_Complete
Read_Clock_Offset_Error

Read_Remote_Supported_Features

Read_Remote_Supported_Features requests a list of the supported features of a remote device.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Read_Remote_Supported_Features_Complete
Read_Remote_Supported_Features_Error

Read_Remote_Version_Information

Read_Remote_Version_Information command will read the values for the version information for the remote Bluetooth device.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Read_Remote_Version_Information_Complete
Read_Remote_Version_Information_Error

Reject_Connection_Request

Reject_Connection_Request is used to decline a new incoming connection request. Execute this command before connection request from another device. By default, all connection requests are accepted.

Command Parameters	Examples	Comments
N/A		

Return Events
Reject_Connection_Request_Complete

Remote_Name_Request

Remote_Name_Request is used to obtain the user-friendly name of another Bluetooth device.

The BD_ADDR command parameter is used to identify the device for which the user-friendly name is to be obtained. The Page_Scan_Repetition_Mode and Page_Scan_Mode command parameters specify the page scan modes supported by the remote device with the BD_ADDR. This is the information that was acquired during the inquiry process. The Clock_Offset parameter is the difference between its own clock and the clock of the remote device with BD_ADDR. Only bits 2 through 16 of the difference are used and they are mapped to this parameter as bits 0 through 14 respectively. A Clock_Offset_Valid_Flag, located in bit 15 of the Clock_Offset command parameter, is used to indicate if the Clock Offset is valid or not.

Command Parameters	Examples	Comments
BD ADDR	0x010203040506	
Page Scan Rep Mode	0x0	0x00=R0; 0x01=R1; 0x02=R2
Page Scan Mode	0x0	0x00=Mandatory Page Scan Mode 0x01=Optional Page Scan Mode I 0x02=Optional Page Scan Mode II 0x03=Optional Page Scan Mode III
Clock Offset	0x0	Bit Format: Bit 14.0 = Bit 16.2 of CLKslave - CLKmaster. Bit 15 = Clock_Offset_ Valid_Flag where 0= Invalid Clock Offset 1=Valid Clock Offset

Return Events
Remote_Name_Request_Complete
Remote_Name_Request_Error

Set_Connection_Encryption

Set_Connection_Encryption is used to enable and disable the link-level encryption.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Encryption_Enable	1	Range: 0 or 1

Return Events
Set_Connection_Encryption_Complete
Set_Connection_Encryption_Error

HCI Link Policy Commands

Exit_Park_Mode

Stops park mode and enters active mode for the specified ACL link.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Mode_Change
Exit_Park_Mode_Error

Exit_Sniff_Mode

Stops Sniff mode and enters active mode for the specified ACL link.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Mode_Change
Exit_Sniff_Mode_Error

Hold_Mode

Places the specified ACL link into Hold Mode.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Max_Interval	0xFFFF	0x0001 - 0xFFFF
Min_Interval	0x01	0x0001 - 0xFFFF

Return Events
Mode_Change
Hold_Mode_Error

Park_Mode

Places the specified ACL link into Park mode.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Beacon_Max_Interval	0xFFFF	0x0001 - 0xFFFF
Beacon_Min_Interval	0x01	0x0001 - 0xFFFF

Return Events
Mode_Change
Park_Mode_Error

QoS_Setup

Used to specify Quality of Service parameters for a connection handle.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

ServiceType	0x01	0=No traffic;
		1=Best effort;
		2=Guaranteed

TokenRate	0	Token rate in Bytes/second
PeakBandwidth	0	Bytes per second
Latency	0xFFFFFFFF	In microseconds
DelayVariation	0xFFFFFFFF	In microseconds

Return Events
Quality_of_Service_Setup_Complete
Quality_of_Service_Setup_Error

Read_Link_Policy_Settings

Reads Link Policy setting for the specified ACL link.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Read_Link_Policy_Settings_Complete
Read_Link_Policy_Settings_Error

Role_Discovery

Description:

Role_Discovery is used for a Bluetooth device to determine which role the device is performing (Master or Slave) for a particular connection.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Role_Discovery_Complete
Role_Discovery_Error

Sniff_Mode

Places the specified ACL link into Sniff mode.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Max_Interval	0xFFFF	0x0001 - 0xFFFF
Min_Interval	0x01	0x0001 - 0xFFFF
Attempt	0x3FF6	0x0001 - 0x7FFF
Timeout	0x7FFF	0x0000 - 0x7FFF

Return Events
Mode_Change
Sniff_Mode_Error

Switch_Role

Switches the current role (master/slave) of the calling device with the role of the device specified.

Command Parameters	Examples	Comments
BD_ADDR	0x010203040506	

Return Events
Role_Change_Complete
Role_Change_Error

Write_Link_Policy_Settings

Writes link policy settings for the specified ACL link.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Link_Policy_Settings	0xF	0x0000: Disable all LM modes 0x0001: Enable master/slave switch 0x0002: Enable Hold Mode 0x0004: Enable Sniff Mode 0x0008: Enable Park Mode 0xF: Enable all (Default)

Return Events
Write_Link_Policy_Settings_Complete
Write_Link_Policy_Settings_Error

HCI Host Controller & Baseband Commands

Change_Local_Name

Change_Local_Name allows the user-friendly name to be modified for the Merlin's Wand.

Command Parameters	Examples	Comments
Name	Merlin's Wand	Maximum string length =32 characters

Return Events
Change_Local_Name_Complete
Change_Local_Name_Error

Delete_Stored_Link_Key

Delete_Stored_Link_Key removes one or all link keys stored in the Merlin's Wand.

Command Parameters	Examples	Comments
BD_ADDR	0x010203040506	
Delete_All_Flag	01	00=Delete only the Link Key for specified BD_ADDR 01=Delete all stored Link Keys

Return Events
Delete_Stored_Link_Key_Complete
Delete_Stored_Link_Key_Error

Host_Buffer_Size

Used by the Merlin's Wand to notify the Merlin's Wand Host Controller about its buffer sizes for ACL and SCO data. The Merlin's Wand Host Controller will segment the data to be transmitted from the Host Controller to Merlin's Wand, so that data contained in HCI Data Packets will not exceed these sizes.

Command Parameters	Examples	Comments
ACL_Data_Length	0x2A0	Only ACL is valid
SCO_Data_Length	0xFF	The value of the Host_SCO_Data_Packet_ Length must be > 399
Total_Num_ACL	10	
Total_Num_SCO	0xFF	

Return Events
Host_Buffer_Size_Complete
Host_Buffer_Size_Error

Read_Authentication_Enable

Read_Authentication_Enable will read the value for the Authentication_Enable parameter, which controls whether Merlin's Wand will require authentication for each connection with other Bluetooth devices.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Authentication_Enable_Complete
Read_Authentication_Enable_Error

Read_Class_of_Device

Read_Class_of_Device will read the value for the Class_of_Device parameter for Merlin's Wand, which is used to indicate its capabilities to other devices.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Class_of_Device_Complete
Read_Class_of_Device_Error

Read_Connection_Accept_Timeout

Read_Connection_Accept_Timeout will read the value for the Connection_Accept_Timeout parameter so that Merlin's Wand can automatically deny a connection request after a specified period has occurred, and to refuse a new connection.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Connection_Accept_Timeout_Complete
Read_Connection_Accept_Timeout_Error

Read_Current_IAC_LAP

Read_Current_IAC_LAP will read the LAP(s) used to create the Inquiry Access Codes (IAC) that Merlin's Wand is simultaneously scanning for during Inquiry Scans.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Current_IAC_LAP_Complete
Read_Current_IAC_LAP_Error

Read_Encryption_Mode

Read_Encryption_Mode will read the value for the Encryption_Mode parameter, which controls whether Merlin's Wand will require encryption for each connection with other Bluetooth devices.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Encryption_Mode_Complete
Read_Encryption_Mode_Error

Read_Link_Supervision_Timeout

Reads link supervision timeout setting for the specified ACL link.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Read_Link_Supervision_Timeout_Complete
Read_Link_Supervision_Timeout_Error

Read_Local_Name

Read_Local_Name reads the stored user-friendly name for Merlin's Wand.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Local_Name_Complete
Read_Local_Name_Error

Read_Number_Of_Supported_IAC

This command will read the value for the number of Inquiry Access Codes (IAC) that Merlin's Wand can simultaneously listen for during an Inquiry Scan.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Number_Of_Supported_IAC_Complete
Read_Number_Of_Supported_IAC_Error

Read_Page_Scan_Mode

Read_Page_Scan_Mode command is used to read the current Page_Scan_Mode of Merlin's Wand.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Page_Scan_Mode_Complete
Read_Page_Scan_Mode_Error

Read_Page_Scan_Period_Mode

Read_Page_Scan_Period_Mode is used to read the Page_Scan_Period_Mode of Merlin's Wand.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Page_Scan_Period_Mode_Complete
Read_Page_Scan_Period_Mode_Error

Read_Page_Timeout

Read_Page_Timeout will read the value for the Page_Reply_Timeout configuration parameter, which allows Merlin's Wand to define the amount of time a connection request will wait for the remote device to respond before the local device returns a connection failure.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Page_Timeout_Complete
Read_Page_Timeout_Error

Read_PIN_Type

Read_PIN_Type will read the PIN type specified in the host controller of Merlin's Wand.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_PIN_Type_Complete
Read_PIN_Type_Error

Read_Scan_Enable

Read_Scan_Enable will read the value for the Scan_Enable configuration parameter, which controls whether or not Merlin's Wand will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Scan_Enable_Complete Read_Scan_Enable_Error

Read_SCO_Flow_Control_Enable

The Read_SCO_Flow_Control_Enable command provides the ability to read the SCO_Flow_Control_Enable setting. By using this setting, Merlin can decide if the Merlin's Wand Host Controller will send Number Of Completed Packets events for SCO HCI_Handles.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_SCO_Flow_Control_Enable_Complete Read_SCO_Flow_Control_Enable_Error

Read_Stored_Link_Key

Read_Stored_Link_Key will read one or all link keys stored in the Merlin's Wand Host Controller.

Command Parameters	Examples	Comments
BD_ADDR	0x010203040506	
Read_All_Flag	01	00=Return Link Key for specified BD_ADDR 01=Return all stored Link Keys

Return Events
Read_Stored_Link_Key_Complete Read_Stored_Link_Key_Error

Read_Voice_Setting

Read_Voice_Setting will read the values for the Voice_Setting parameter in Merlin's Wand, which controls all the various settings for the voice connections.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Voice_Setting_Complete Read_Voice_Setting_Error

Reset

Resets the Bluetooth Host Controller, Link Manager, and the radio module of Merlin's Wand. After executing this command, the application has to be restarted.

Command Parameters	Examples	Comments
N/A		Will reset to default values for the parameters

Return Events
Reset_Complete

Set_Event_Filter

Set_Event_Filter is used by the Host to specify different event filters. The Host may issue this command multiple times to request various conditions for the same type of event filter and for different types of event filters.

Command Parameters	Examples	Comments
FilterType	0x00	0x00=Clear All Filters 0x01=Inquiry Result 0x02=Connection Setup
FilterConditionType	0x00	0x00=New device responded to the Inquiry process 0x01= Device with a specific Class of Device responded to the Inquiry process. 0x02=Device with specific BD_ADDR responded to the Inquiry process.

Command Parameters	Examples	Comments
Condition	0x01	0x00=Allow Connections from all devices 0x01=Allow Connections from a device with a specific Class of Device 0x02=Allow Connections from a device with a specific BD_ADDR

Return Events
Set_Event_Filter_Complete
Set_Event_Filter_Error

Set_Event_Mask

Set_Event_Mask is used to control which events are generated by the HCI for the Host.

Command Parameters	Examples	Comments
Event_Mask		NO_EVENTS 0x0000 INQUIRY_RESULT 0x0001 INQUIRY_COMPLETE 0x0002 INQUIRY_CANCELED 0x0004 LINK_CONNECT_IND 0x0008 SCO_CONNECT_IND 0x0010 LINK_DISCONNECT 0x0020 LINK_CONNECT_CNF 0x0040 LINK_CON_RESTRICT 0x0080 MODE_CHANGE 0x0100 ACCESSIBLE_CHANGE 0x0200 AUTHENTICATED 0x0400 ENCRYPTION_CHANGE 0x0800 SECURITY_CHANGE 0x1000 ROLE_CHANGE 0x2000 SCO_DISCONNECT 0x4000 SCO_CONNECT_CNF 0x8000 ALL_EVENTS 0xffff

Return Events
Set_Event_Mask_Complete
Set_Event_Mask_Error

Write_Authentication_Enable

This command will write the value for the Authentication_Enable parameter, which controls whether Merlin's Wand will require authentication for each connection with other Bluetooth devices.

Command Parameters	Examples	Comments
Authentication_Enable	0x0	0x00=Authentication disabled. Default 0x01=Authentication enabled for all connection

Return Events
Write_Authentication_Enable_Complete
Write_Authentication_Enable_Error

Write_Class_of_Device

Write_Class_of_Device will write the value for the Class_of_Device parameter, which is used to indicate its capabilities to other devices.

Command Parameters	Examples	Comments
CoD	0x000000	Class of Device for the device

Return Events
Write_Class_of_Device_Complete
Write_Class_of_Device_Error

Write_Connection_Accept_Timeout

Write_Connection_Accept_Timeout will write the value for the Connection_Accept_Timeout configuration parameter, which allows Merlin's Wand to automatically deny a connection request after a specified period has occurred, and to refuse a new connection.

Command Parameters	Examples	Comments
Timeout	0x00	Connection Accept Timeout measured in Number of Baseband slots. Interval Length = N * 0.625 msec (1 Baseband slot) Range for N: 0x0001 – 0xB540 Time Range: 0.625 msec - 29 seconds Default: N = 0x1FA0 Time = 5 Sec

Return Events
Write_Connection_Accept_Timeout_Complete
Write_Connection_Accept_Timeout_Error

Write_Current_IAC_LAP

Will write the LAP(s) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans.

Command Parameters	Examples	Comments
IAC_LAP	0x9E8B33	0x9E8B00-0x9E8B3F
IAC_LAP		0x9E8B00-0x9E8B3F
IAC_LAP		0x9E8B00-0x9E8B3F
IAC_LAP		0x9E8B00-0x9E8B3F
IAC_LAP		0x9E8B00-0x9E8B3F
IAC_LAP		0x9E8B00-0x9E8B3F

Return Events
Write_Current_IAC_LAP_Complete
Write_Current_IAC_LAP_Error

Write_Encryption_Mode

Write_Encryption_Mode command will write the value for the Encryption_Mode parameter, which controls whether Merlin's Wand will require encryption for each connection with other Bluetooth devices.

Command Parameters	Examples	Comments
Encryption Mode	0x0	0x00=Encryption disabled. Default 0x01=Encryption only for point-to-point packets 0x02=Encryption for both point-to-point and broadcast packets

Return Events
Write_Encryption_Mode_Complete
Write_Encryption_Mode_Error

Write_Link_Supervision_Timeout

Writes link supervision timeout setting for the specified ACL link.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Timeout	0x7D00	0x0001 - 0xFFFF

Return Events
Write_Link_Supervision_Timeout_Complete
Write_Link_Supervision_Timeout_Error

Write_Page_Timeout

Write_Page_Timeout command will write the value for the Page_Reply_Timeout configuration parameter, which allows Merlin's Wand to define the amount of time a connection request will wait for the remote device to respond before the local device returns a connection failure.

Command Parameters	Examples	Comments
Timeout	0x10	0=Illegal Page Timeout. Must be larger than 0 N = 0xXXXX Page Timeout measured in Number of Baseband slots. Interval Length = N * 0.625 msec (1 Baseband slot) Range for N: 0x0001 – 0xFFFF Time Range: 0.625 msec -40.9 Seconds Default: N = 0x2000 Time = 5.12 Sec

Return Events
Write_Page_Timeout_Complete Write_Page_Timeout_Error

Write_PIN_Type

Write_PIN_Type will specify whether the Host supports variable PIN or only fixed PINs.

Command Parameters	Examples	Comments
PIN_Type	0	0x00=Variable PIN 0x01=Fixed PIN

Return Events
Write_PIN_Type_Complete Write_PIN_Type_Error

Write_Scan_Enable

The Write_Scan_Enable command will write the value for the Scan_Enable configuration parameter into Merlin's Wand, which controls whether or not Merlin's Wand will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices.

Command Parameters	Examples	Comments
Scan_Enable	3	0x00=No Scans enabled. 0x01=Inquiry Scan enabled Page Scan disabled. 0x02=Inquiry Scan disabled. Page Scan enabled. 0x03=Inquiry Scan enabled Page Scan enabled. (Default)

Return Events
Write_Scan_Enable_Complete
Write_Scan_Enable_Error

Write_Stored_Link_Key

Write_Stored_Link_Key command will write a link key to the Merlin's Wand host controller.

Command Parameters	Examples	Comments
BD_ADDR	0x010203040506	
Link_Key	0x01020304	

Return Events
Write_Stored_Link_Key_Complete
Write_Stored_Link_Key_Error

Write_Voice_Setting

The Write_Voice_Setting command will write the values for the Voice_Setting parameter into Merlin's Wand, which controls all the various as settings for the voice connections.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Voice_Setting	0x0062	0x0060=CVSD coding 0x0061=u-Law coding 0x0062=A-law coding

Return Events
Write_Voice_Setting_Complete
Write_Voice_Setting_Error

HCI Informational Commands

Read_BD_ADDR

Read_BD_ADDR will read the value of Merlin's Wand's address. The BD_ADDR is a 48-bit unique identifier for a Bluetooth device.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_BD_ADDR_Complete
Read_BD_ADDR_Error

Read_Buffer_Size

Read_Buffer_Size returns the size of the HCI buffers in Merlin's Wand. These buffers are used by Merlin's Wand's Host Controller to buffer data that is to be transmitted.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Buffer_Size_Complete
Read_Buffer_Size_Error

Read_Country_Code

Read_Country_Code command will read the value for the Country_Code status parameter in Merlin's Wand. The Country_Code defines which range of frequency band of the ISM 2.4 GHz band will be used by the device.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Country_Code_Complete
Read_Country_Code_Error

Read_Local_Supported_Features

Read_Local_Supported_Features will request a list of the supported features for Merlin's Wand.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Local_Supported_Features_Complete Read_Local_Supported_Features_Error

Read_Local_Version_Information

Read_Local_Version_Information command will read the values for the version information for Merlin's Wand.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Local_Version_Information_Complete Read_Local_Version_Information_Error

HCI Testing Commands

Enable_Device_Under_Test_Mode

The Enable_Device_Under_Test_Mode command will allow Merlin's Wand to enter test mode via LMP test commands. Merlin's Wand issues this command when it wants to become the DUT for the Testing scenarios as described in the Bluetooth Test Mode.

Command Parameters	Examples	Comments
N/A		

Return Events
Enable_Device_Under_Test_Mode_Complete Enable_Device_Under_Test_Mode_Error

Read_Loopback_Mode

Read_Loopback_Mode will read the value for the setting of the Merlin's Wand Host Controller's Loopback_Mode. The setting of the Loopback_Mode will determine the path of information.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Loopback_Mode_Complete
Read_Loopback_Mode_Error

Write_Loopback_Mode

The Write_Loopback_Mode will write the value for the setting of the Host Controller's Loopback_Mode into Merlin's Wand.

Command Parameters	Examples	Comments
Loopback_Mode	0	0x00=No Loopback mode enabled. Default 0x01=Enable Local Loopback 0x02=Enable Remote Loopback

Return Events
Write_Loopback_Mode_Complete
Write_Loopback_Mode_Error

CATC-Specific HCI Commands

CATC_BER_Test

This command will measure Bit Error Rate (BER) when fully loaded DH1, DH3, DH5, DM1, DM3 or DM5 packets are sent from master to slave on the link.

Command Parameters	Examples	Comments
HCI_Handle		
Number_Of_Packets	0000	0x0000=Unlimited number of packets will be sent 0x0001=0xFFFF number of packets will be sent

Command Parameters	Examples	Comments
BER_Packet_Type	03	0x00=DH1 0x01=DH3 0x02=DH5 0x03=DM1 0x04=DM3 0x05=DM5
Test_Data_Type	00	0x00=Send PBRS (same as in Bluetooth test mode) 0x01=Every octet that is sent equals Test_Data
Test_Data	FF	Data to send
BER_Interval	10	A packet is sent every BER_Interval frame

Return Events
CATC_BER_Test_Complete
CATC_BER_Test_Error

CATC_Change_Headset_Gain

Controls the gain of the microphone or speaker of the headset.

Command Parameters	Examples	Comments
Device	"Speaker"	"Microphone" or "Speaker" ("Speaker" is default)
Gain	0	0x00 - 0x0F

Return Events
CATC_Change_Headset_Gain_Complete
CATC_Change_Headset_Gain_Error

CATC_Read_Headset_Gain

Reads the gain of the microphone or speaker of the headset.

Command Parameters	Examples	Comments
Device	"Speaker"	"Microphone" or "Speaker" ("Speaker" is default)

Return Events
CATC_Read_Headset_Gain_Complete
CATC_Read_Headset_Gain_Error

CATC_Read_PIN_Response_Enable

Read the global flag for allowing PINs to be sent to requesting devices during authentication.

Command Parameters	Examples	Comments
NA		

CATC_Read_Revision_Information

Merlin's Wand uses this command to read the revision number of the Merlin's Wand baseband controller.

Command Parameters	Examples	Comments
N/A		

Return Events
CATC_Read_Revision_Information_Complete
CATC_Read_Revision_Information_Error

CATC_Self_Test

This command will perform self-test of Merlin's Wand.

Command Parameters	Examples	Comments
N/A		

Return Events
CATC_Self_Test_Complete
CATC_Self_Test_Error

CATC_Set_Default_PIN_Code

Set the default PIN code, used for authentication purposes whenever a device-specific PIN is unavailable.

Command Parameters	Examples	Comments
PIN_Code		

CATC_Write_PIN_Response_Enable

Write the global flag for allowing PINs to be sent to requesting devices during authentication.

Command Parameters	Examples	Comments
PIN_Response_Enable		0 = disable 1 = enable

Return Events
Command Complete: Will return after the operation has completed.

A.2 Other HCI Events

Events
Command_Complete
PIN_Code_Request
Paring_Complete
Encryption_Change
Disconnect_Complete
Link_Key_Request_Complete

A.3 L2CAP Command Descriptions

ConfigurationResponse

Response to an incoming configuration request.

Command Parameters	Examples	Comments
Reason	"Accept"	"Accept" (Default) "Reject" "Reject - unacceptable params" "Reject - unknown options"
TokenRate	0x00	
TokenBucketSize	0x00	
PeakBandwidth	0x00	
Latency	0xFFFFFFFF	
DelayVariation	0xFFFFFFFF	

Return Events
ConfigurationResponse_Complete

ConfigurationRequest

Sets L2CAP connection options.

Command Parameters	Examples	Comments
ServiceType	0x01	
TokenRate	0x00	
TokenBucketSize	0x00	
PeakBandWidth	0x00	
Latency	0xFFFFFFFF	
DelayVariation	0xFFFFFFFF	

Return Events
ConfigurationSetup_Complete
Error

ConnectRequest

Requests establishment of an L2CAP channel in the remote Bluetooth device.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
PSM	0x1001	
Receive MTU	0x1B6	

Return Events
Connection_Complete
Connection_Failed

ConnectResponse

Indicates the response to the incoming connection request.This command should be executed before Connection Request.

Command Parameters	Examples	Comments
Response		Accept (Default) Reject_Pending Reject_PSM_Not_Supported Reject_Security_Block Reject_No_Resources

Return Events
ConnectResponse_Complete

DeregisterPsm

Deregisters the specified PSM.

Command Parameters	Examples	Comments
PSM	0x1001	

Return Events
DeregisterPsm_Complete
DeregisterPsm_Failed

DisconnectRequest

Requests the disconnection of the specified L2CAP channel.

Command Parameters	Examples	Comments
CID	0x0040	

Return Events
Disconnection_Complete
Disconnection_Failed

EchoRequest

Sends an Echo Request over the L2CAP channel.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Data	"echo"	

Return Events
EchoRequest_Complete
EchoRequest_Failed

InfoRequest

Sends an Info Request over the L2CAP channel. Info requests are used to exchange implementation-specific information regarding L2CAP's capabilities.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
InfoRequest_Complete
InfoRequest_Failed

RegisterPsm

Registers a PSM identifier with L2CAP. Once registered, the protocol can initiate connections and data transfers as well as receive connection requests and data.

Command Parameters	Examples	Comments
PSM	0x1001	
Receive MTU	0x1B6	

Return Events
RegisterPsm_Complete
RegisterPsm_Failed

SendData

Sends data on the specified L2CAP channel.

Command Parameters	Examples	Comments
CID	0x0040	
Data Pipe	"Pipe1"	Data_Pipe should be created in the Data Transfer Manager.

Return Events
SendData_Complete
SendData_Failed

A.4 Other L2CAP Events

Events
Connection_Indication
Disconnection_Indication
Data_Indication
Write_Configuration_Complete
Command_Complete

A.5 SDP Command Descriptions

AddProfileServiceRecord

This command will add a pre-defined Service Record according to one of the Bluetooth wireless technology profiles to the SDP database.

Command Parameters	Examples	Comments
Profile	HeadSet	The following are values of the Profile parameter: Cordless Dialup Fax FileTransfer GN HCRP_Client HCRP_Server Headset HeadsetAudioGateway Intercom LAN NAP ObjectPush PANU Printing SerialPort Sync SyncCommand
ServerChannel	0x01	Server channel has to be entered for all profiles except for InterCom and Cordless.

Return Events
AddProfileServiceRecord_Complete
AddProfileServiceRecord_Error

AddServiceRecord

This command will add a pre-defined Service Record according to one of the Bluetooth wireless technology profiles to the SDP database.

Command Parameters	Examples	Comments
Filename	"C:/Records.sdp"	Click the "..." button to choose a file. Choosing a file will automatically load the records within that file, and those record names will be in the drop-down for Record Name.

Command Parameters	Examples	Comments
Record Name	"FTP Test Record"	Record Names loaded from the Filename file will be in the drop-down for this parameter.
ServerChannel		To leave the server channel as is, enter 0 or leave this blank.

Return Events
AddServiceRecord_Complete
AddServiceRecord_Failed

ProfileServiceSearch

This command will search for support of one of the Bluetooth wireless technology profiles.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Profile	HeadSet	The following are values of the Profile parameter: Cordless Dialup Fax FileTransfer GN HCRP_Client HCRP_Server Headset HeadsetAudioGateway Intercom LAN NAP ObjectPush PANU Printing SerialPort Sync SyncCommand

Return Events
ProfileServiceSearch_Complete
ProfileServiceSearch_Failed

RequestServiceAttribute

This command will retrieve specific attribute values from a specific service record. The Service Record Handle from a specific Service Record and a list of AttributeIDs to be retrieved from that Service Record are supplied as parameters. Up to three AttributeIDs can be searched in one request. Service Record Handle is usually retrieved by using RequestServiceSearch command.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
ServiceRecordHandle	0x00010000	
AttributeID	0x1108	You can specify between 0 and 3 AttributeIDs.
AttributeID	0x1203	
AttributeID		

Return Events
RequestServiceAttribute_Response
Search_Failed

RequestServiceSearch

This command will locate Service Records that match the ServiceSearch Pattern of Service Class IDs. The SDP server will return all Service Record Handles of Service Records that match the given Service Search Pattern. Up to three ServiceClassIDs can be searched in one request.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
ServiceClassID	0x1204	Between 0 and 3 Service Class IDs can be specified.
ServiceClassID	0x1110	
ServiceClassID	0x1000	

Return Events
RequestServiceSearch_Response
Search_Failed

RequestServiceSearchAttribute

This command combines the capabilities of the RequestServiceAttribute and RequestServiceSearch into a single request. This command will retrieve all Attribute values that match the ServiceSearch pattern.

Command Parameters	Examples	Comments
HCI_Handle		
ServiceClassID		Note that all Attributes are requested since a range of 0x0000-0xFFFF is specified by default.
ServiceClassID		
ServiceClassID		

Return Events
RequestServiceSearchAttribute_Response
Search_Failed

ResetDatabase

This command will remove all Service Records from the database in Merlin's Wand.

Command Parameters	Examples	Comments
N/A		

Return Events
ResetDatabase_Complete
ResetDatabase_Failed

A.6 RFCOMM Command Descriptions

AcceptChannel

This command will accept or reject incoming request to open an RFCOMM channel from RFCOMM server. This command should be executed before RFCOMM connection request from another device. By default, all connection requests are accepted.

Command Parameters	Examples	Comments
Accept	TRUE	(Values: TRUE/FALSE)

Return Events
AcceptChannel_Complete

AcceptPortSettings

This command will accept or reject PortSettings received during RequestPortSettings event. This command should be executed before PortSettings request from another device. By default, all requests are accepted.

Command Parameters	Examples	Comments
Accept	TRUE	(Values: TRUE/FALSE)

Return Events
AcceptPortSettings_Complete

AdvanceCredit

Advances a specified number of credits to a particular RFCOMM connection.

Command Parameters	Examples	Comments
(HCI / DLCI)	(0x0001, 0x02)	
Credit	20	Number of credits to advance

Return Events
AdvanceCredit_Complete
AdvanceCredit_Error

CloseClientChannel

This command will close an established RFCOMM channel between Merlin's Wand and a remotely connected device.

Command Parameters	Examples	Comments
(HCI/DLCI)	(0x0001, 0x02)	The DLCI value is returned by the OpenClientChannel command

Return Events
CloseClientChannel_Complete
CloseClientChannel_Error

CreditFlowEnabled

This command is used to check if credit-based flow control has been negotiated for the current RFCOMM session.

Command Parameters	Examples	Comments
(HCI/DLCI)	(0x0001, 0x02)	The DLCI value is returned by the OpenClientChannel command

Return Events
CreditFlowEnabled_Complete

DeregisterServerChannel

Deregisters an RFCOMM server channel.

Command Parameters	Examples	Comments
ServerChannel	0x01	ServerChannel must first be registered via RegisterServerChannel command.

Return Events
DeregisterServerChannel_Complete
DeregisterServerChannel_Error

OpenClientChannel

This command will open an RFCOMM channel.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
ServerChannel	0x01	Range: 1-30
MaxFrameSize	0x7F	Range: 23-32767 Default is 127
Credit	0x20	The number of frames that the sender has available
AttachTo		Command_Generator Virtual_COM_Port

Return Events
OpenClientChannel_Complete
OpenClientChannel_Failed

RegisterServerChannel

This command will register ServerChannel with an RFCOMM server so that the server can respond to incoming OpenClientChannel requests.

Command Parameters	Examples	Comments
AttachTo		Command_Generator Virtual_COM_Port

Return Events
RegisterServerChannel_Complete
RegisterServerChannel_Error

RequestPortSettings

This command will request a change to the current PortSettings.

Command Parameters	Examples	Comments
(HCI/DLCI)	(0x0001, 0x02)	For values, see below.
BaudRate	9600	
DataFormat	0x02	
FlowControl	0x00	
Xon	0x11	
Xoff	0x13	

Return Events
RequestPortSettings_Complete
RequestPortSettings_Failed

Parameter Values:

- BaudRate: Specifies the baud rate. Note that the baud rate setting does not actually affect RFCOMM throughput.

  Values: 2400, 4800, 7200, 9600, 19200, 38400, 57600, 115200, 230400

- DataFormat: The following values identify the number of data bits.

  Values:

  0x00 -- DATA_BITS_5
  0x02 -- DATA_BITS_6
  0x01 -- DATA_BITS_7
  0x03 -- DATA_BITS_8

- Flow Control:

  Values:

0x00 -- FLOW_CTRL_NONE
0x01 -- XON_ON_INPUT
0x02 -- XON_ON_OUTPUT
0x04 -- RTR_ON_INPUT
0x08 -- RTR_ON_OUTPUT
0x10 -- RTC_ON_INPUT
0x20 -- RTC_ON_OUTPUT
Xon:

Value: Default Xon char -- 0x11

- Xoff:

  Value: Default Xoff char -- 0x13

RequestPortStatus

This command will request the status of the PortSettings for the remote device.

Command Parameters	Examples	Comments
(HCI/DLCI)	(0x0001, 0x02)	

Return Events
RequestPortStatus_Complete
RequestPortStatus_Error

SendData

Causes Merlin's Wand to send data by pipe value to remote device over the specified channel.

Command Parameters	Examples	Comments
(HCI/DLCI)	(0x0001, 0x02)	
Data Pipe	Pipe1	Data Pipe should be created in the Data Transfer Manager

Return Events
SendData_Complete
SendData_Failed

SendTest

This command causes Merlin's Wand to sent a test frame to a remote device over the specified channel.

Command Parameters	Examples	Comments
(HCI/DLCI)	(0x0001, 0x02)	

Return Events
SendTest_Complete
SendTest_Failed

SetLineStatus

This command will send the LineStatus command to the remote device. It allows the RFCOMM user to communicate overrun framing and parity errors to the remote device.

Command Parameters	Examples	Comments
(HCI/DLCI)	(0x0001, 0x02)	LineStatus Values:
LineStatus	0x0F	0x01 -- Set to indicate an error. 0x02 -- Set to indicate an overrun error. 0x04 -- Set to indicate a parity error. 0x08 -- Set to indicate a framing error.

Return Events
SetLineStatus_Complete
SetLineStatus_Failed

SetModemStatus

This command will send the ModemStatus to the remote device. It allows the user to send Flow Control and V.24 signals to the remote device.

Command Parameters	Examples	Comments
(HCI/DLCI)	(0x0001, 0x02)	
ModemSignals	0x8C	Modem Signal Values: 0x8c - Ready to communicate, ready to receive, and data valid 0x02 (FLOW) - Set when sender is unable to receive frames 0x04 (RTC) - Set when sender is ready to communicate. 0x08 (RTR) - Set when sender is ready to receive data. 0x40 (IC) - Set when a call is incoming. 0x80 (DV) - Set when valid data is being sent.

Command Parameters	Examples	Comments
BreakLength	0x00	0-15 indicates the length of the break signal in 200ms units. 0 indicates no break signal.

Return Events
SetModemStatus_Complete
SetModemStatus_Failed

SendATCommand

This command will send a selected AT command.

Command Parameters	Examples	Comments
(HCI/DLCI)	(0x0001, 0x02)	
AT_Command	RING	Values: AT + CKPD = 200: Headset Button pressed. AT + VGM = 1: Microphone gain. +VGM = 1: Microphone gain 1. RING OK ERROR BUSY CONNECT NO_CARRIER NO_DIAL_TONE

Return Events
Send_AT_Command_Complete
Send_AT_Command_Error

A.7 Other RFCOMM Events

Events
OpenClientChannel_Request
CloseClientChannel_Indication
Data_Indication
PortNegotiation_Indication
RequestPortStatus_Indication
ModemStatus_Indication

Events
LineStatus_Indication
Flow_Off_Indication
Flow_On_Indication

A.8 TCS Command Descriptions

RegisterIntercomProfile

Registers an Intercom identifier with TCS. Once registered, the protocol can initiate connections as well as receive connection requests.

Command Parameters	Examples	Comments
N/A		

Return Events
Register_Intercom_Profile_Complete
Register_Intercom_Profile_Error

Open_TCS_Channel

This command opens an L2CAP channel with TCS PSM and initializes a TCS state machine into NULL state.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Open_TCS_Channel_Complete
Open_TCS_Channel_Failed

Start_TCS_Call

This command must be called right after TCSOpenChannel. It automatically sends a sequence of TCS messages according to the Intercom profile specification of the TCS state machine. After successful execution of this command, TCS state machine is in ACTIVE state and SCO connection is opened.

Command Parameters	Examples	Comments
N/A		

Return Events
Start_TCS_Call_Complete
Start_TCS_Call_Error

Disconnect_TCS_Call

This command is called to close an existing TCS connection according to the Intercom profile specification of the TCS state machine, close the L2CAP connection, and close the SCO connection.

Command Parameters	Examples	Comments
N/A		

Return Events
Disconnect_TCS_Call_Complete
Disconnect_TCS_Call_Error

Send_Info_Message

This command can be called after a TCS channel is opened. It sends an INFORMATION TCS message with a called party number.

Command Parameters	Examples	Comments
Phone_Number	408 727 6600	Phone number may contain up to 10 digits.

Return Events
Send_Info_Complete
Send_Info_Error

A.9 OBEX Command Descriptions

ClientAbort

This command will abort a ClientGet or ClientPut operation.

Command Parameters	Examples	Comments
N/A		

Return Events

ClientConnect

This command will create an OBEX connection with a remote device.

Command Parameters	Examples	Comments
BD_ADDR	0x010203040506	An HCI Connection has to be established before calling this command.

Return Events
ClientConnect_Complete
ClientConnect_Error

ClientDisconnect

This command will cause the remote device to close the established OBEX channel.

Command Parameters	Examples	Comments
N/A		

Return Events
ClientDisconnect_Complete
ClientDisconnect_Error

ClientGet

This command will initiate an OBEX Get operation in the remote device for the object named in the store handle. This operation is only valid over an OBEX connection.

Command Parameters	Examples	Comments
Object	"VCard.vcf"	

Return Events
ClientGet_Complete
ClientGet_Error

ClientPut

This command will initiate an OBEX Put operation in the remote device for the object defined in the FileName parameter.

Command Parameters	Examples	Comments
Filename	"C:\VCard.vcf"	

Return Events
ClientPut_Complete
ClientPut_Error

ClientSetPath

This command will initiate an OBEX SetPath operation in the remote device. Flags indicate SetPath option such as Backup.

Command Parameters	Examples	Comments
Path	"C:\OBEX"	
Flags	0x00	Bit 0: Back up a level before applying name (equivalent to ../ on many systems) Bit 1: Don't create an directory if it not exist. Returns an error instead.

Return Events
OBEX_Command_Complete
ClientSetPath_Error

ServerDeinit

This command will deinitialize the OBEX server.

Command Parameters	Examples	Comments
N/A		

Return Events
ServerDeinit_Complete
ServerDeinit_Error

ServerInit

This command will initialize the OBEX server.

Command Parameters	Examples	Comments
N/A		

Return Events
ServerInit_Complete
ServerInit_Error

ServerSetPath

Sets the path where received OBEX files are stored.

Command Parameters	Examples	Comments
Path	"C:\OBEX"	Use the "..." button to select a path, or type one in.

Return Events
ServerSetPath_Complete
ServerSetPath_Event
ServerSetPath_Error

A.10 BNEP Command Descriptions

Accept

Command Parameters	Examples	Comments
AcceptConnection		

Open

Command Parameters	Examples	Comments
BD_ADDR		.

Close

Command Parameters	Examples	Comments
BNEP_ADDR		

SetUpConnectionReq

Command Parameters	Examples	Comments
BNEP_ADDR		.
Destination UUID		
Source UUID		

SentPKT

Command Parameters	Examples	Comments
BNEP_ADDR		.
Packet_Type		

SendControlPKT

Command Parameters	Examples	Comments
BNEP_ADDR		
Control Packet Type		
Range Start		
Range End		

RegisterBNEP

Command Parameters	Examples	Comments
NA		

DeregisterBNEP

Command Parameters	Examples	Comments
NA		

SetControlTimeout

Command Parameters	Examples	Comments
BNEP_ADDR		
Timeout		

Appendix B: Command Generator Examples

This chapter provides fourteen Command Generator examples. These examples consist of command sequences that are presented in order to illustrate useful scenarios. Please note that these examples do not cover all possible alternatives.

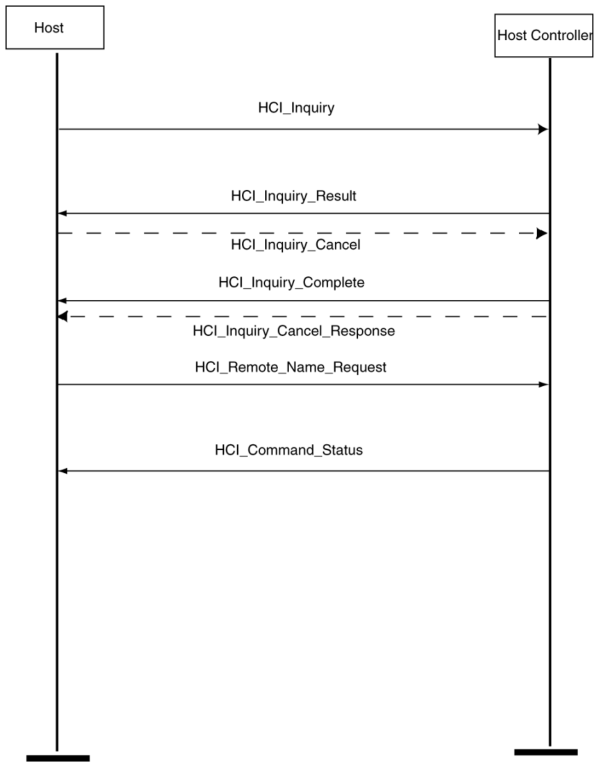
- Device Discovery and Remote Name Request
- Establish Baseband Connection
- Baseband Disconnection
- Create Audio Connection
- Establish L2CAP Connection
- L2CAP Channel Disconnect
- SDP Profile Service Search
- SDP Reset Database and Add Profile Service Record
- RFCOMM Client Channel Establishment
- RFCOMM Client Channel Disconnection
- RFCOMM Register Server Channel and Accept Incoming Connection
- Establish TCS Connection
- OBEX Server Init and Accept Incoming Connection
- OBEX Client Connection and Client Get & Put

Each example is illustrated with a diagram that shows communications between a host and host controller.

Notation used in this chapter:

- **Hexagon** = Condition needed to start the transaction.
- **Solid Arrow** represents a command.
- **Dashed Arrow** represents optional command.
- **Host** = Merlin's Wand application
- **Host Controller** = Merlin's Wand device

B.1 Device Discovery and Remote Name Request



Procedure

In this scenario, Merlin's Wand performs a General Inquiry and a Remote Name Request.

Step 1 Select HCI tab.

Step 2 Select **Inquiry** from the menu. You can use the default settings for the Inquiry_Length (8 seconds) and Num_Responses (10).

Step 3 Click **Execute.**

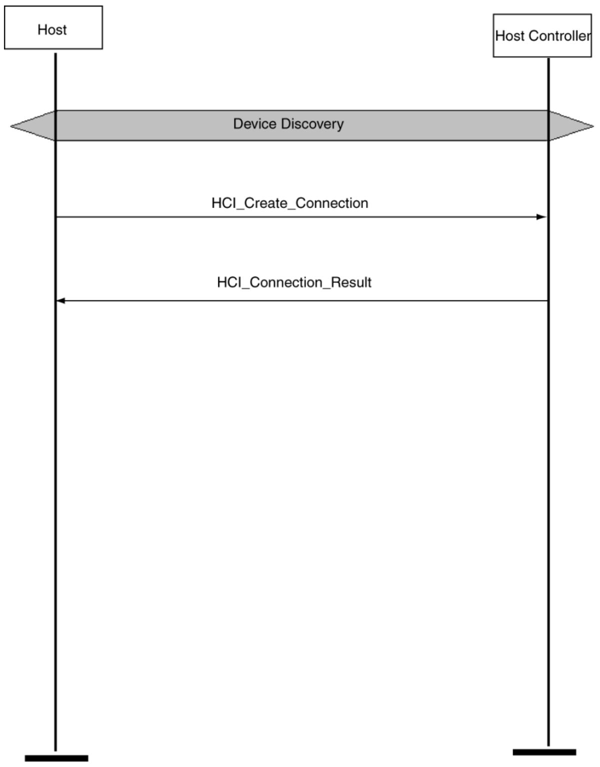
The Event Log should display an Inquiry_Result for each found device followed by an Inquiry_Complete event.

Step 4 Select **Remote_Name_Request** from the menu. Select the target device from the BD ADDR drop-down menu. You can use the default settings for the other drop-down menus.

Step 5 Click **Execute**.

The target device should then respond to the command with its name.

B.2 Establish Baseband Connection



Procedure

In this scenario, Merlin's Wand creates a Baseband (ACL) Connection.

This procedure assumes that Device Discovery has already been performed. See "Device Discovery and Remote Name Request" on page 138.

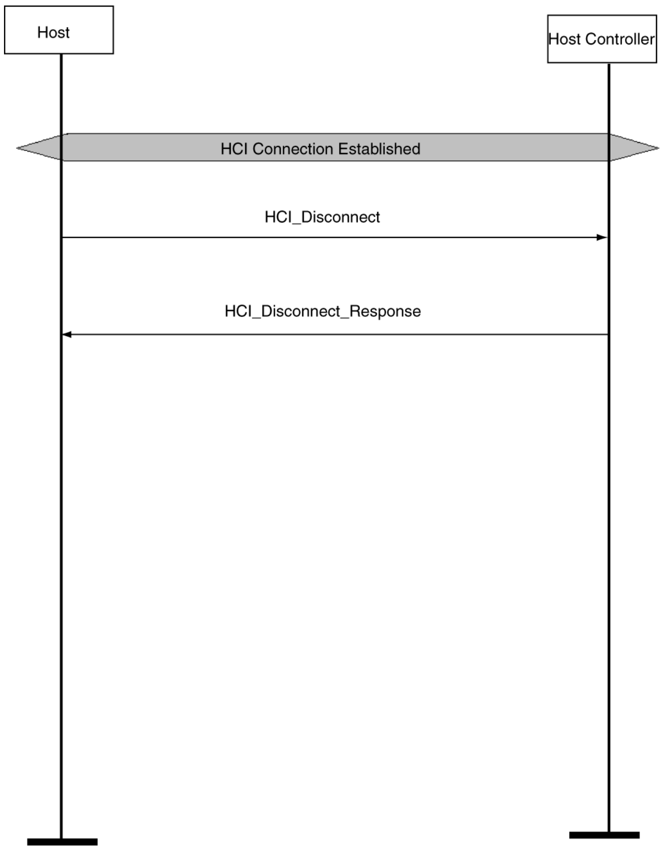
Step 1 From the HCI menu select **Create_Connection**.

Step 2 Select the target device from the BD_ADDR drop-down menu or enter a new BD_ADDR.

Step 3 Click **Execute**.

The Event Log should display a Connection_Complete or Connection_Failed response.

B.3 Baseband Disconnection



Procedure

In this scenario, Merlin's Wand terminates a Baseband connection. These steps continue the connection you established in the preceding scenario.

This procedure assumes that an ACL connection exists. See "Establish Baseband Connection" on page 139.

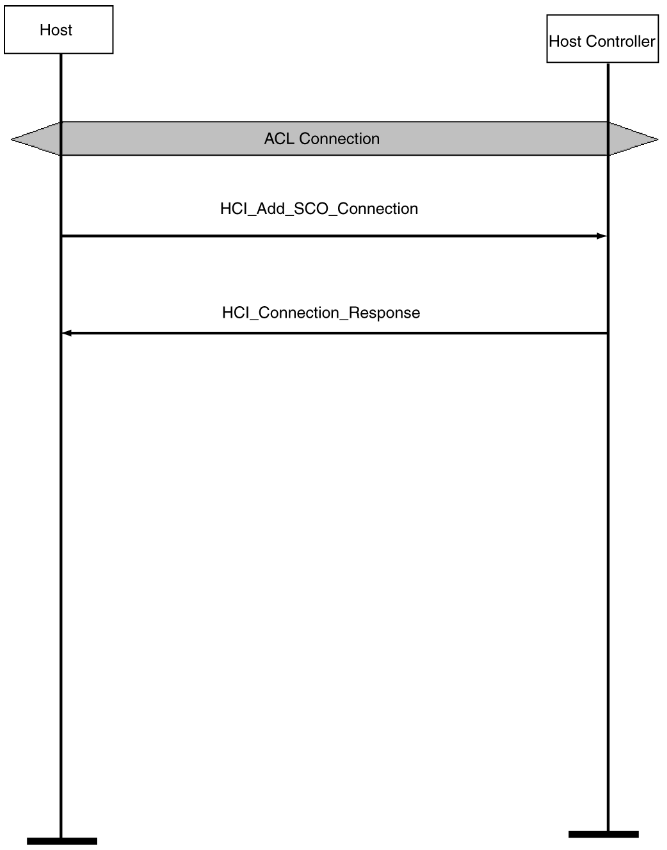
Step 1 From the HCI menu, select **Disconnect**.

Step 2 From the HCI_Handle drop-down menu, select a handle.

Step 3 Click **Execute**.

For each Disconnect, you should see a return event in the Event Log that indicates the outcome of the command.

B.4 Create Audio Connection



Procedure

In this scenario, Merlin's Wand creates an SCO connection.

This procedure assumes that you have established an ACL connection between Merlin's Wand and the target device. See "Establish Baseband Connection" on page 139.

Step 1 From the HCI menu, select Add_SCO_Connection from the menu.

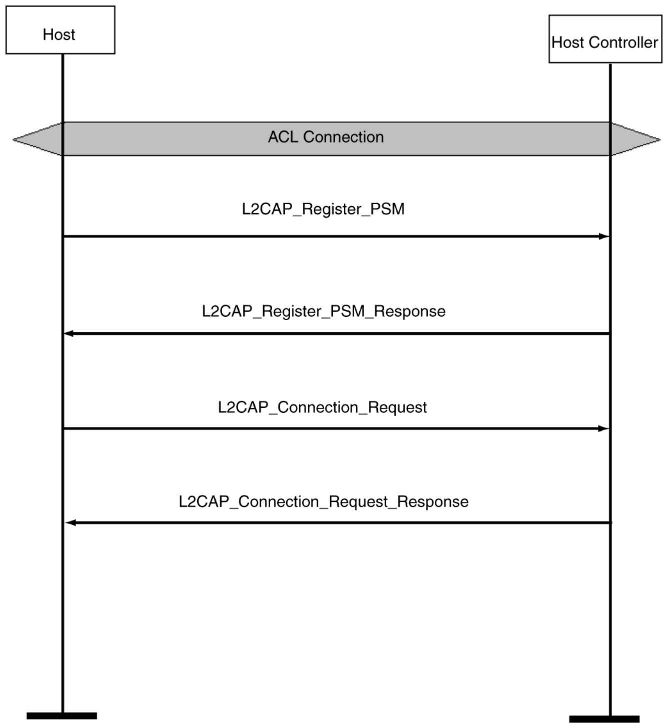
Step 2 Select the HCI_Handle for the previously established Baseband connection (for example, 0x0001) from the HCI_Handle parameter drop-down menu.

Step 3 Select a packet type from the Packet Type parameter drop-down menu.

Step 4 Click **Execute**.

The Event Log should indicate that the command was executed and that the target device responded with "Add_SCO_Connection_Complete" or "Add_SCO_Connection_Error."

B.5 L2CAP Connection



Procedure

In this scenario, Merlin's Wand establishes an L2CAP connection.

This procedure assumes that an ACL connection has been established. See "Establish Baseband Connection" on page 139.

Step 1 Click the L2CAP tab to display the L2CAP drop-down menu.

Step 2 Select **Register_PSM** from the menu.

Merlin's Wand must register its PSM channel before it can form an L2CAP connection.

Step 3 Select or type a PSM from the PSM menu.

Step 4 Select or type the Receive MTU from the Receive MTU menu (default value can be used).

Step 5 Click **Execute**.

Step 6 Repeat this procedure for the target device. The target device must also select a PSM.

Event Log should register "RegisterPsm_Complete."

Step 7 Select ConnectRequest from the L2CAP menu.

Step 8 Select an HCI Handle from the HCI_Handle drop-down menu.

To determine which HCI_Handle value is correct, open the **Piconet** window on the far left side of the Merlin's Wand application.

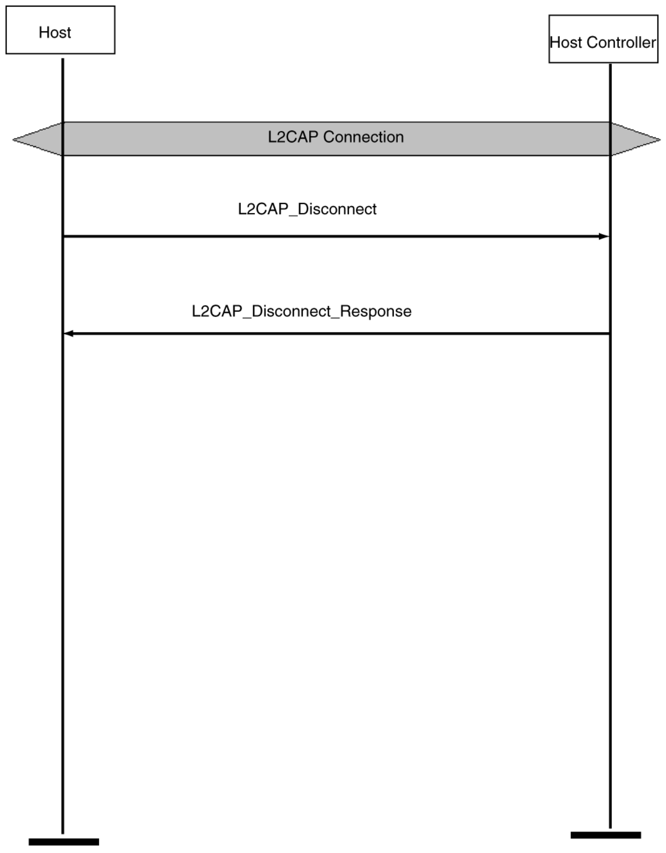
Step 9 Select or type a PSM from the PSM menu.

Step 10 Select or type the Receive MTU from the Receive MTU menu (default value can be used).

Step 11 Click **Execute**.

The Event Log should indicate that the command was executed and that a connection has been established.

B.6 L2CAP Channel Disconnect



Procedure

In this scenario, Merlin's Wand terminates an L2CAP connection.

This procedure assumes that an L2CAP connection has been established. See "L2CAP Connection" on page 142.

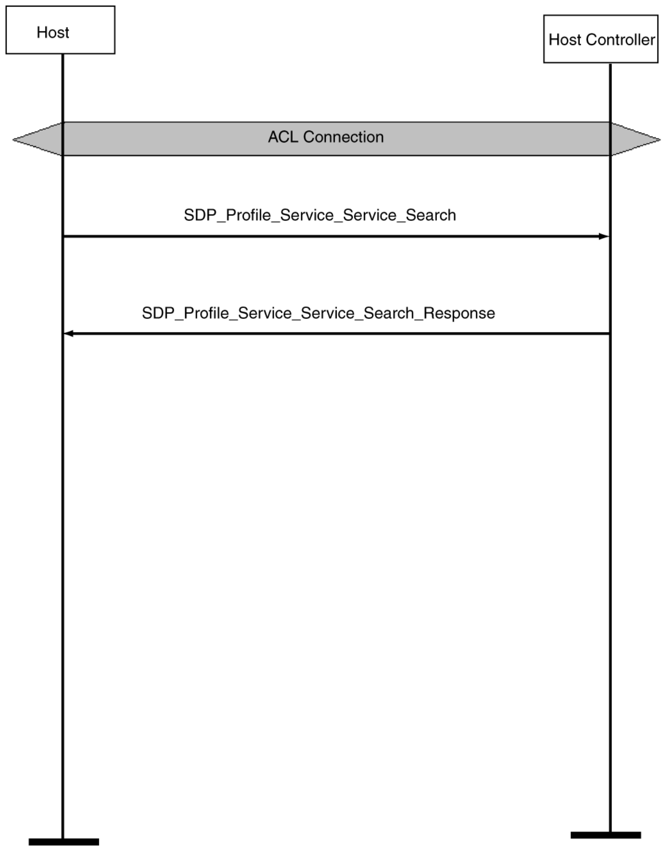
Step 1 From the L2CAP menu, select DisconnectRequest.

Step 2 Select the appropriate CID from the CID menu.

Step 3 Click **Execute**.

The Event Log should indicate that the command was successfully completed, with "Disconnection_Complete."

B.7 SDP Profile Service Search


Host
Host Controller
ACL Connection
SDP_Profile_Service_Service_Search
SDP_Profile_Service_Service_Search_Response

Procedure

In this scenario, Merlin's Wand conducts a Profile Service Search.

This procedure assumes that an ACL connection has been established. "Establish Baseband Connection" on page 139.

Step 1 Click the SDP tab to display the SDP menu.

Step 2 Select ProfileServiceSearch from the menu.

Step 3 Select an HCI Handle from the HCI_Handle drop-down list.

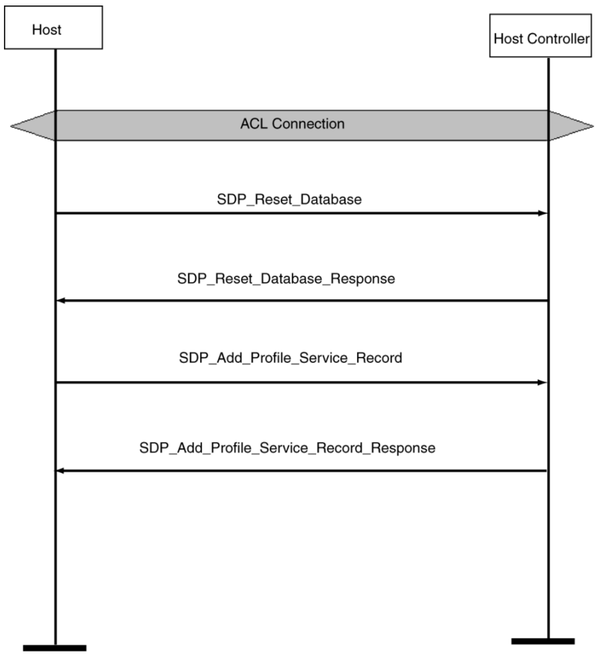
You can determine the HCI Handle from the **Piconet** window.

Step 4 Select a profile from the Profile menu.

Step 5 Click **Execute**.

The Event Log should return "ProfileServiceSearch_Complete," as well as the results of the search.

B.8 SDP Reset Database and Add Profile Service Record


Host
Host Controller
ACL Connection
SDP_Reset_Database
SDP_Reset_Database_Response
SDP_Add_Profile_Service_Record
SDP_Add_Profile_Service_Record_Response

Procedure

In this scenario, Merlin's Wand resets the SDP database and then adds an SDP Profile Service Record.

This procedure assumes that an ACL connection has been established between Merlin's Wand and the target device. "Establish Baseband Connection" on page 139.

Note A connection is not necessary to perform a Reset_Database or AddProfileServiceRecord.

Step 1 From the SDP tab select **ResetDatabase**.

Step 2 Click **Execute**.

The Event Log should indicate that the database was reset.

Step 3 Select **AddProfileServiceRecord** from the menu.

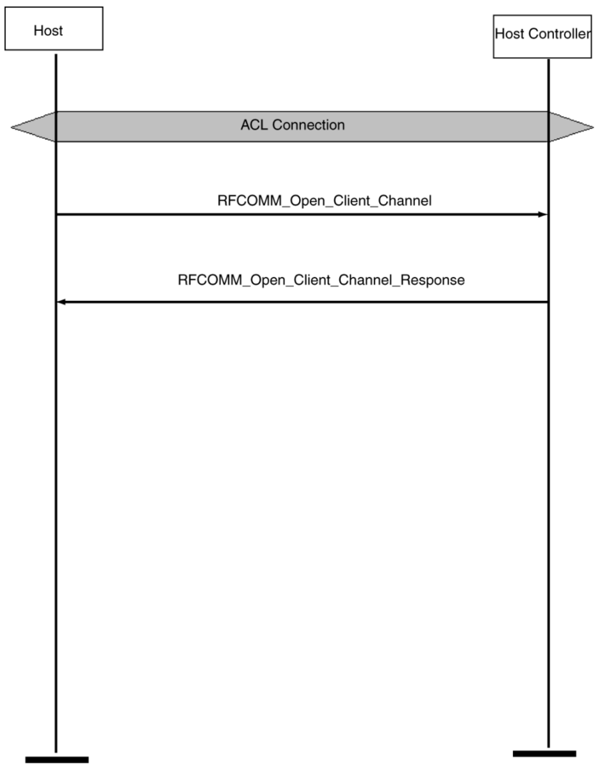
Step 4 Select a profile from the Profile menu.

Step 5 Select a server channel from the Channel menu.

Step 6 Click **Execute**.

Success will be indicated in the Event Log with "AddProfileServiceRecord_Complete" and the type of profile added.

B.9 RFCOMM Client Channel Establishment


Host
Host Controller
ACL Connection
RFCOMM_Open_Client_Channel
RFCOMM_Open_Client_Channel_Response

Procedure

In this scenario, Merlin's Wand opens an RFCOMM client channel.

This procedure assumes that an ACL connection has been established and that the target device has assumed the role of an RFCOMM server. See "Establish Baseband Connection" on page 139.

Step 1 Click the RFCOMM tab to open the RFCOMM drop-down menu.

Step 2 Select **OpenClientChannel** from the menu.

Step 3 Select an HCI Handle from the HCI_Handle drop-down list.

Step 4 Select a Server Channel from the ServerChannel menu.

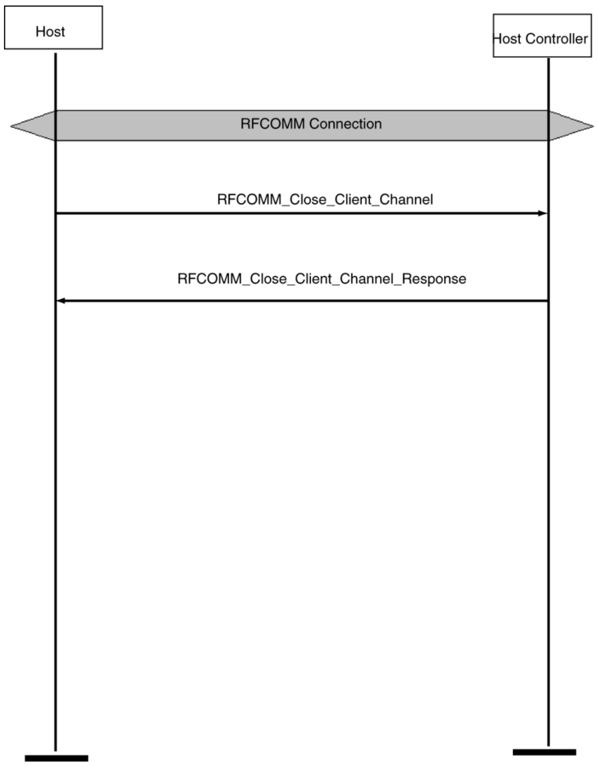
Step 5 Select a Max Frame Size from the MaxFrameSize menu.

Step 6 Select the number of credits from the Credit menu.

Step 7 Click **Execute**.

"OpenClientChannel_Complete" in the Event Log indicates that a client channel was successfully opened.

B.10 RFCOMM Client Channel Disconnection


Host
Host Controller
RFCOMM Connection
RFCOMM_Close_Client_Channel
RFCOMM_Close_Client_Channel_Response

Procedure

In this scenario, Merlin's Wand closes an RFCOMM client channel.

This procedure assumes that an RFCOMM channel has been established. See "RFCOMM Client Channel Establishment" on page 147.

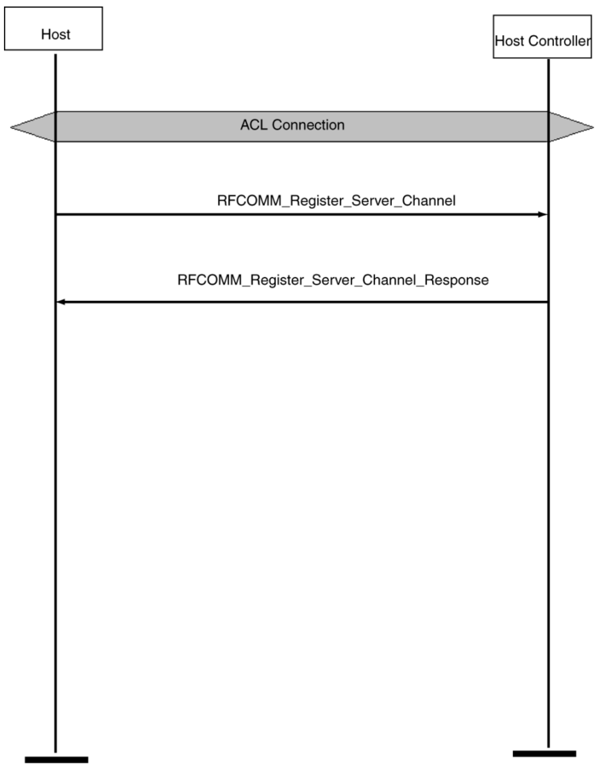
Step 1 From the RFCOMM menu select **CloseClientChannel**.

Step 2 Select the HCI/DLCI combination from the (HCI/DLCI) menu.

Step 3 Click **Execute**.

The Event Log should indicate a response to the command such as "CloseClientChannel_Complete."

B.11 RFCOMM Register Server Channel


Host
Host Controller
ACL Connection
RFCOMM_Register_Server_Channel
RFCOMM_Register_Server_Channel_Response

Procedure

In this scenario, Merlin's Wand registers a Server Channel.

This procedure assumes that an ACL connection has been established. See "Establish Baseband Connection" on page 139.

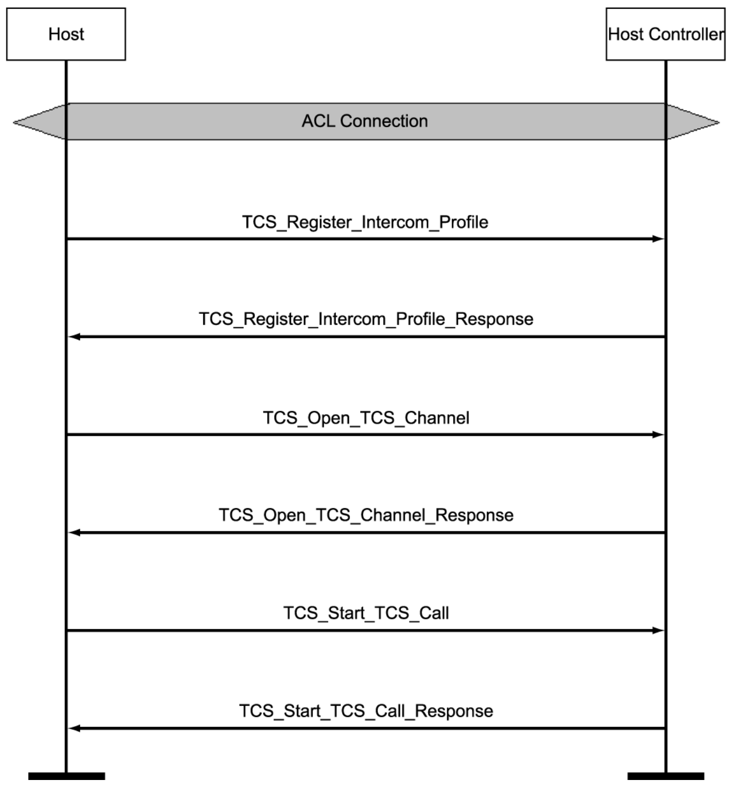
Note A connection is not necessary to call a RegisterServerChannel command.

Step 1 From the RFCOMM menu select **RegisterServerChannel**.

Step 2 Click **Execute**.

The Event Log should indicate successful completion of the command with the response "RegisterServerChannel_Complete." On completion of these steps, the application is ready to accept incoming RFCOMM connections.

B.12 Establish TCS Connection


Host
Host Controller
ACL Connection
TCS_Register_Intercom_Profile
TCS_Register_Intercom_Profile_Response
TCS_Open_TCS_Channel
TCS_Open_TCS_Channel_Response
TCS_Start_TCS_Call
TCS_Start_TCS_Call_Response

Procedure

In this scenario, Merlin's Wand establishes a TCS connection.

This procedure assumes that an ACL connection has been established. See "Establish Baseband Connection" on page 139.

Step 1 Click the **TCS tab** to display the TCS drop-down menu.

Step 2 Select **Register_Intercom_Profile** from the menu.

Merlin 's Wand must register its Intercom profile before it can form a TCS connection.

Step 3 Click **Execute**.

The Event Log should display "Register_Intercom_Profile_Complete."

Step 4 Repeat Steps 1-3 for the target device.

Step 5 Select **Open_TCS_Channel** from the menu and select an HCI handle.

Merlin 's Wand must create an ACL connection before it can form a TCS connection.

Step 6 Click **Execute**.

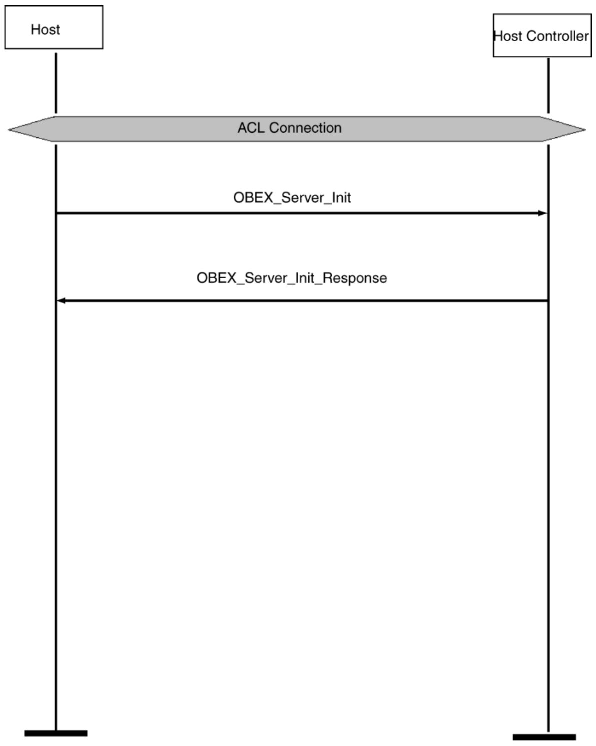
Event Log should display "Open_TCS_Channel_Complete."

Step 7 Select **Start_TCS_Call** from the menu.

Step 8 Click **Execute**.

Event Log should display "Start_TCS_Call_Complete."

B.13 OBEX Server Init



Procedure

In this scenario, Merlin's Wand initializes itself as an OBEX server. This scenario assumes that an ACL connection exists. See "Establish Baseband Connection" on page 139.

Note A connection is not necessary to call an OBEX ServerInit function.

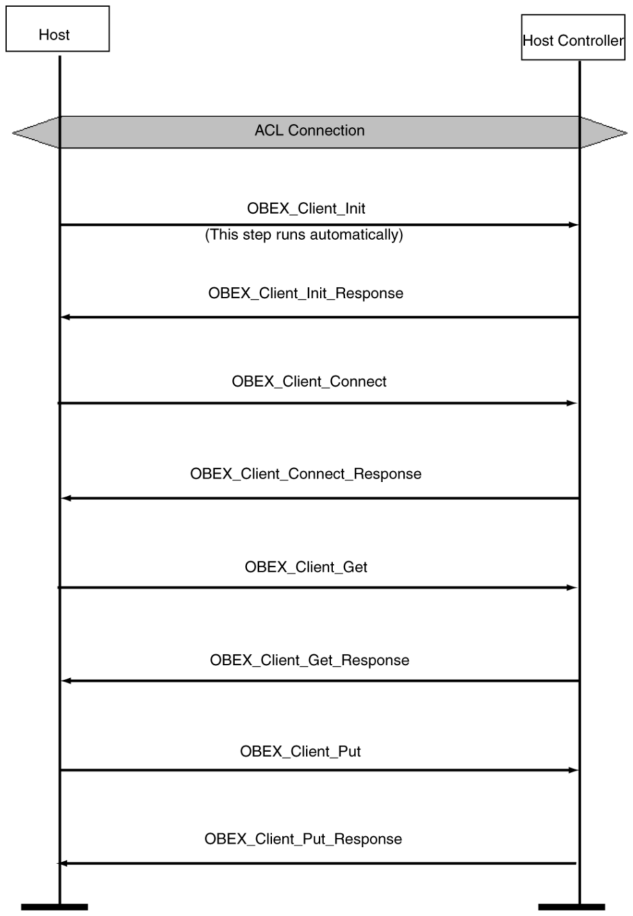
Step 1 Click the OBEX tab to display the OBEX menu.

Step 2 Select **ServerInit** from the menu.

Step 3 Click **Execute**.

On completion of these steps, the application is ready to accept an incoming OBEX connection.

B.14 OBEX Client Connection and Client Get & Put



Procedure

In this scenario, Merlin's Wand forms a client connection with the target device and then retrieves a text file from the target and sends one to it.

This procedure assumes that an ACL connection has been established (see "Establish Baseband Connection" on page 139). It also assumes that the target device has been configured as an OBEX server.

Note: When the OBEX window is first opened, Merlin's Wand will automatically run an OBEX_ClientInit command and initiate itself as an OBEX client. This means that you do not have to manually execute a ClientInit command at the start of this procedure.

Step 1 Click the OBEX tab to display the OBEX menu.

Step 2 Select **ClientConnect** from the menu.

Step 3 Select the target device from the **BD_ADDR** parameter menu.

Step 4 Click **Execute**.

The Event Log should indicate that a connection was established.

Step 5 Select **ClientGet** from the command menu.

Step 6 Type in the name of a file that is to be transferred from the Server into the **Object** parameter box.

Step 7 Click **Execute**.

The Event Log should indicate that the file was transferred. A Save As dialog box should open.

Step 8 Enter a name for the file you are retrieving. Select a directory, then click **Save**.

Step 9 Select **ClientPut** from the command menu.

Step 10 In the box marked **Filename,** type the name of a file that is to be transferred to the server, or use the browse ... button to locate the file you want to put. The Open dialog will come up, allowing you to navigate to the desired file.

Step 11 Click **Execute**.

The Event Log should indicate that the file was transferred.

Appendix C: Merlin's Wand Scripting Commands

Merlin's Wand supports scripting commands to help automate testing processes and commonly used sequences of Bluetooth commands. Custom scripts can be written, saved, and run in Script Manager.

C.1 Bluetooth Addresses

Bluetooth addresses are represented in scripts as binary strings in big-endian byte order. For example, the Bluetooth address "0x010203040506" would be represented in the script as:

```
DeviceAddress = '010203040506';
```

Comparisons can be performed using binary strings. For example:

```
if ( DeviceAddress == '010203040506' )
{
   #do something based on comparison here
}
```

C.2 Basic Commands

Main()

```
Main()
```

Parameter	Meaning	Default Value	Comments
N/A			

Return value

None.

Comments

This is the entry point into a script. When a script is run, the script's Main() function will be called. Include this command at the beginning of every script.

Example

```
Main()
{
        #include body of script here
    }
```

Clock()

`Clock()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

The number of milliseconds that have elapsed since the system was started.

Comments

This function returns the amount of time that the system has been running. It can be used to find out how long it takes to run a script or a series of commands within a script.

Example

```
time1 = Clock();
# Put script commands here
time2 = Clock();
Trace("Elapsed time is ", time2-time1, "\n");
```

Connect()

`Connect(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to connect with		

Return value

- "Success"
- "Already connected"
- "Timed out"
- "Failed: Disconnection in progress"
- "Failure"

Comments

Establishes an ACL connection with the specified device

Example

```
result = Connect(Devices[0]);
if(result != "Success")
{
   MessageBox("Failed to connect!");
}
```

Disconnect()

`Disconnect(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to connect with		

Return value

- "Success"
- "Failure"
- "Timed out"

Comments

Closes the ACL connection with the specified device

Example

```
result = Disconnect(Devices[0]);
if(result != "Success")
{
   MessageBox("Failed to disconnect!");
}
```

DoInquiry()

`DoInquiry(IAC, Timeout)`

Parameter	Meaning	Default Value	Comments
IAC	Inquiry Access Code	GIAC	"GIAC", or a 32-bit integer value
Timeout	Timeout in units of 1.2 seconds	5	

Return value

Array of Bluetooth addresses that were found during the inquiry.

Comments

Calling DoInquiry() will block for the duration specified by *Timeout*. The function returns an array of devices that were found during the inquiry. These can be addressed individually.

The current version of Merlin's Wand hardware only supports GIAC inquiries.

Example

```
# Use default parameters
Devices = DoInquiry();
Trace("First device was: ", Devices[0]);
```

GetDeviceClass()

`GetDeviceClass()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- Class of device
- "Failure"

Comments

Returns the current device class of Merlin's Wand

Example

```
Trace("Merlin's Wand device class: ", GetDeviceClass());
```

GetRemoteDeviceName()

`GetRemoteDeviceName(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		

Return value

- Device name
- "Not connected"
- "Failure"

Comments

Queries the specified device for its name.

An ACL connection must be established before calling GetRemoteDevice-Name().

Example

```
Trace("Device ", Devices[0], "is named ",
GetRemoteDeviceName(Devices[0]));
```

MessageBox()

`MessageBox(Message, Caption)`

Parameter	Meaning	Default Value	Comments
Message	Text to display in the message box		
Caption	Caption of the message box	"Script Message"	

Return value

None.

Comments

Bring up a simple message box function with one "OK" button. This is a good way to pause execution of the script or indicate errors.

Example

```
MessageBox("Failed to connect", "Connection Failure");
```

SetDeviceClass()

`SetDeviceClass(Class)`

Parameter	Meaning	Default Value	Comments
Class	Device class for Merlin's Wand		Device class is a 3-byte value

Return value

- "Success"
- "Failure"

Comments

Sets the device class of Merlin's Wand

Example

```
SetDeviceClass(0x010203);
```

`Sleep()`

`Sleep(Time)`

Parameter	Meaning	Default Value	Comments
Time	Time in ms		

Return value

None.

Comments

Delays program execution for Time in milliseconds.

Example

```
Sleep(1000); # Sleep for one second
```

C.3 Pipe Commands

`ClosePipe()`

`ClosePipe(PipeName, PipeType)`

Parameter	Meaning	Default Value	Comments
PipeName	Name of the data pipe to open		
PipeType	"Transmit" or "Receive" pipe	"Receive"	

Return value

- "Success"

- "Failure"
- "Pipe Not Found"
- "Invalid parameter"

Comments

Closes the specified data pipe.

Example

```
ClosePipe("Data1", "Receive");
```

DeletePipe()

```
DeletePipe(PipeName, PipeType)
```

Parameter	Meaning	Default Value	Comments
PipeName	Name of the data pipe to open		
PipeType	"Transmit" or "Receive" pipe	"Receive"	

Return value

- "Success"
- "Invalid parameter"
- "Pipe not found"

Comments

Removes a pipe from the Data Transfer Manager pipe list. In the case of "Receive" pipes, all data associated with the pipe is lost. "Transmit" pipes will only be removed from the Data Transfer Manager list.

Example

```
DeletePipe("Data1", "Receive");
```

OpenPipe()

```
OpenPipe(PipeName, PipeType)
```

Parameter	Meaning	Default Value	Comments
PipeName	Name of the data pipe to open		

Parameter	Meaning	Default Value	Comments
PipeType	"Transmit" or "Receive" pipe	"Receive"	

Return value

- "Success"
- "Failure"
- "Pipe Not Found"

Comments

Opens a data pipe for reading or writing. If the data pipe type is "Receive" and the pipe does not exist, a new pipe will be created. All open pipes will be automatically closed upon script termination.

Example

```
OpenPipe("Data1", "Receive");
```

ReadPipe()

```
ReadPipe(PipeName, PipeType, ByteCount)
```

Parameter	Meaning	Default Value	Comments
PipeName	Name of the data pipe to open		
PipeType	"Transmit" or "Receive" pipe		
ByteCount	Number of bytes to read		1-65535

Return values

Returns a list with three values: *result, bytes read,* and *data*.

Result (element 0) is one of the following:

- "Success"
- "Failure"
- "Invalid parameter"
- "Pipe not found"
- "Pipe not open"

Bytes read (element 1) is the number of bytes read in this transaction. Valid only if result is "Success".

Data (element 2) is the raw data received in the transaction. Valid only if result is "Success".

Comments

Reads the specified amount of data from an open pipe.

Example

```
result = ReadPipe("Data1", "Receive", 1024);
if(result[0] == "Success")
{
   Trace("Read ", result[1], "bytes:\n");
   Trace(result[2]);
}
```

WritePipe()

`WritePipe(PipeName, Data)`

Parameter	Meaning	Default Value	Comments
PipeName	Name of the data pipe to open		
Data	Data to write to the pipe		This can be a string, integer, list or raw data.

Return value

- "Success"
- "Failure"
- "Pipe not found"
- "Pipe not open"
- "Not supported"

Comments

Writes data to the specified pipe. Note that only "Receive" pipes can be written to.

Example

```
result = WritePipe("Data1", "This is a string written to a
pipe");
result = WritePipe("Data1", '3C7EFFFF7E3C');
result = WritePipe("Data1", 0x01020304);
```

C.4 HCI Commands

`HCIAcceptConnectionRequest()`

`HCIAcceptConnectionRequest()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"

Comments

Sets the accept connection request variable to True.

Example

```
status = HCIAcceptConnectionRequest();
```

`HCIAddSCOConnection()`

`HCIAddSCOConnection(Address, Type)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to connect with		
Type	Type of SCO connection to establish	["HV3"]	A list of one or more of the following strings: "DM1", "HV1", "HV3" or "DV"

Return value

- "Success"
- "Not connected"
- "Not supported"
- "Failure"

Comments

Attempts to establish an SCO connection with the specified device.

An ACL connection must already be established with the device before calling HCIAddSCOConnection.

Example

```
result = HCIAddSCOConnection(Devices[0], ["DM1", "HV1"]);
if(result != "Success")
{
   MessageBox(result, "Failed to add SCO connection!");
}
```

HCIAuthenticationRequested()

`HCIAuthenticationRequested(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to authenticate with		A connection must exist with this device for an authentication request to work.

Return value

- "Success"
- "Failure"
- "Failed: Device not found"
- "Timed Out"
- "Not connected"

Comments

Attempts to authenticate an existing link with the specified device.

Example

```
result = HCIAuthenticationRequested (Devices[0]);
if(result != "Success")
{
   MessageBox(result, "Failed to authenticate!");
}
```

HCICatcBerTest()

`HCICatcBerTest(Address, NumberOfPackets, BERPacketType, TestDataType, TestData, BERInterval)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of the remote device		

Parameter	Meaning	Default Value	Comments
NumberOf Packets	The number of baseband packets to be transmitted		0x0000 - an unlimited number of packets 0x0001 - 0xFFFF - number of packets
BERPacket Type			0x00 - DH1 0x01 - DH3 0x02 - DH5 0x03 - DM1 0x04 - DM3 0x05 - DM5
TestDataType			0x00 - send Bluetooth test mode PRBS 0x01 - every octet equals TestData
TestData			Data to send
BERInterval	A packet is sent every BERInterval frame		

Return value

- "Success"
- "Failure"
- "Not connected"

Comments

This command is used to measure BER when fully loaded baseband packets are sent from master to slave on the link.

Example

```
Trace("Test Control : ",
HCICatcTestControl(Address,1,1,2,2,4), "\n");
Trace("Enter Test Mode : ", HCICatcEnterTestMode(Address),
"\n");
Trace("BER Test : ", HCICatcBerTest(Address,1,3,0,0xFF,10),
"\n");
```

HCICatcChangeHeadsetGain()

`HCICatcChangeHeadsetGain(Device, Gain)`

Parameter	Meaning	Default Value	Comments
Device	Speaker or microphone		Values: "Speaker", "Microphone"

Parameter	Meaning	Default Value	Comments
Gain	New gain of the device		Values: 0 – 0xF

Return value

- "Success"
- "Failure"
- "Not connected"
- "No SCO connection"

Comments

This command is used to change gain of connected speaker or microphone. In order to use this command, an SCO connection must exist.

Example

```
Main()
{
   result = Connect('00803713BDF0');
   Trace("Connection result : ", result, "\n");
   if( result == "Success")
   {
      result = HCIAddSCOConnection( '00803713BDF0',
["HV1"]);
      Trace("SCO Connection result : ", result, "\n");
      if( result == "Success")
      {
         index = 0;
         while(index < 16)
         {
            result = HCICatcChangeHeadsetGain("Speaker",
index);
            Trace("Change speaker gain: ", result, "\n");
            result = HCICatcReadHeadsetGain("Speaker",
index);
            Trace("Read speaker gain: ", result, "\n");
            index = index + 1;
            Sleep(2000);
         }
         index = 0;
         while(index < 16)
         {
            result =
HCICatcChangeHeadsetGain("Microphone", index);
            Trace("Change microphone gain: ", result,
"\n");
            result = HCICatcReadHeadsetGain("Microphone");
            Trace("Read microphone gain : ", result, "\n");
            index = index + 1;
            Sleep(2000);
         }
         status = HCIRemoveSCOConnection('00803713BDF0');
         Trace("SCO disconnect result: ", status, "\n");
      }
      status = Disconnect('00803713BDF0');
      Trace("Disconnect result: ", status, "\n");
   }
}
```

HCICatcReadHeadsetGain()

`HCICatcReadHeadsetGain(Device)`

Parameter	Meaning	Default Value	Comments
Device	Speaker or microphone		Values: "Speaker", "Microphone"

Return values

Returns a list with two values: *status* and *gain*.

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Not found"
- "No SCO connection"

Gain (element 1) is the one-byte value of the headset gain. Range is 0 to 15.

Comments

This command is used to read current gain of connected speaker or microphone. In order to use this command, an SCO connection must exist.

Example

See the example for the HCICatcChangeHeadsetGain command.

CATC_Read_PIN_Response_Enable()

Read the global flag for allowing PINs to be sent to requesting devices during authentication.

Command Parameters	Examples	Comments
NA		

HCICatcReadRevisionInformation()

`HCICatcReadRevisionInformation()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *revision*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Revision (element 1) is the Merlin's Wand revision information.

Comments

This command returns the information about the current firmware.

Example

```
Revision = HCICatcReadRevisionInformation();
if( Revision[0] == "Success")
Trace("Merlin's Wand Revision Info : ", Revision[1], "\n");
```

HCICatcSelfTest()

`HCICatcReadRevisionInformation()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

This command is used to perform a self test on a local device.

Example

```
Trace("Merlin's Wand Self Test : ", HCICatcSelfTest(),
"\n");
```

C.0.1 CATC_Write_PIN_Response_Enable

Write the global flag for allowing PINs to be sent to requesting devices during authentication.

Command Parameters	Examples	Comments
PIN_Response_Enable		0 = disable 1 = enable

Return Events
Command Complete: Will return after the operation has completed.

HCIChangeConnectionLinkKey()

`HCIChangeConnectionLinkKey(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of the remote		

Return value

- "Success"
- "Failure"
- "Not found"
- "Not connected"

Comments

This command is used to force both devices associated with a connection to generate a new link key.

Example

```
result = HCIChangeConnectionLinkKey('00803713BDF0');
Trace("Change Connection Link Key: ", result, "\n");
```

HCIChangeConnectionPacketType()

`HCIChangeConnectionPacketType(Address, PacketType)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to connect with		
PacketType			"DH1" "DH3" "DH5" "DM1" "DM3" "DM5" "AUX1" "HV1" "HV2" "HV3" "DV"

Return value

- "Success"
- "Failure"
- "Not found"
- "Not connected"
- "Not supported"

Comments

This command is used to change which baseband packet type can be used for a connection

Example

```
result = HCIChangeConnectionPacketType('00803713BDF0',
["DM3","DM5"]);
Trace("Change Connection Packet Type:\n");
Trace(" Status            ",   result[0], "\n");
```

`HCIChangeLocalName()`

`HCIChangeLocalName(Name)`

Parameter	Meaning	Default Value	Comments
Name	String that contains the new name for the local device		

Return value

- "Success"
- "Failure"

Comments

Attempts to change the name of the local device.

Example

```
result = HCIChangeLocalName("Joe's Device");
if(result != "Success")
{
   MessageBox(result, "Failed to change name!");
}
```

`HCIDeleteStoredLinkKey()`

`HCIDeleteStoredLinkKey(Address, DeleteAll)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device that will have its link key deleted		
DeleteAll	Boolean value that indicates whether to delete only the specified address's link key, or all link keys	0	0 or 1

Return value

- "Success"

- "Failure"

Comments

Attempts to delete the stored link key for the specified address or for all addresses, depending on the value of DeleteAll.

Example

```
result = HCIDeleteStoredLinkKey('6E8110AC0008', 1);
if(result != "Success")
{
   MessageBox(result, "No link keys were deleted.");
}
```

HCIEnableDeviceUnderTestMode()

`HCIEnableDeviceUnderTestMode()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

This command will allow the local Bluetooth device to enter a test mode via LMP test commands

Example

```
result = HCIEnableDeviceUnderTestMode();
Trace("Enabled DUT : ", result, "\n");
```

HCIExitParkMode()

`HCIExitParkMode(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		

Return value

- "Success"

- "Failure"
- "Failed: Device not found"
- "Not connected"

Comments

Switches the current role of the device in the piconet.

Example

```
Device = '010203040506';
result = HCIExitParkMode(Device);
Trace("HCIExitParkMode result is: ", result, "\n");
```

HCIExitSniffMode()

HCIExitSniffMode(Address)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		

Return value

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Comments

Exits Sniff mode.

Example

```
Device = '010203040506';
result = HCIExitSniffMode(Device);
Trace("HCIExitSniffMode result is: ", result, "\n");
```

HCIHoldMode()

`HCIHoldMode(Address, MaxInterval, MinInterval)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		
MaxInterval	Maximum number of 0.625-msec intervals to wait in Hold mode.		Range is 0x0001 to 0xFFFF (0.625 msec to 40.9 sec).
MinInterval	Minimum number of 0.625-msec intervals to wait in Hold mode.		Range is 0x0001 to 0xFFFF (0.625 msec to 40.9 sec).

Return value

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Comments

Enters Hold mode with parameters as specified.

Example

```
Device = '010203040506';
result = HCIHoldMode(Device, 0xFFFF, 0x50);
Trace("HCIHoldMode result is: ", result, "\n");
```

HCIMasterLinkKey()

`HCIMasterLinkKey(KeyFlag)`

Parameter	Meaning	Default Value	Comments
KeyFlag			0x0 use semi-permanent link keys 0x1 use temporary link keys

Return values

Returns a list with three values: *status, HCI handle,* and *key flag.*

Status (element 0) is one of the following:

- "Success"
- "Failure"

HCI handle (element 1) is the handle for the ACL connection.

Key flag (element 2) is the key flag (either 0 or 1).

Comments

This command is used to force the master of the piconet to use temporary or semi-permanent link keys.

Example

```
result = HCIMasterLinkKey(0);
Trace("Merlin's Wand MasterLinkKey returned:", result[0],
"\n");
if(result[0] == "Success")
{
   Trace(" Connection Handle : 0x", result[1], "\n");
   Trace(" Key Flag          : 0x", result[2], "\n");
}
```

HCIParkMode()

HCIParkMode(Address, BeaconMaxInterval, BeaconMinInterval)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		
Beacon MaxInterval	Maximum number of 0.625-msec intervals between beacons.		Range is 0x0001 to 0xFFFF (0.625 msec to 40.9 sec).
Beacon MinInterval	Minimum number of 0.625-msec intervals between beacons.		Range is 0x0001 to 0xFFFF (0.625 msec to 40.9 sec).

Return value

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Comments

Enters Park mode with parameters as specified.

Example

```
Device = '010203040506';
result = HCIParkMode(Device, 0xFFFF, 0x100);
Trace("HCIParkMode result is: ", result, "\n");
```

HCIPINCodeRequestNegativeReply()

`HCIPINCodeRequestNegativeReply(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device for which no PIN code will be supplied.		

Return value

- "Success"
- "Failure"

Comments

Specifies a device for which no PIN code will be supplied, thus causing a pair request to fail.

Example

```
result = HCIPINCodeRequestNegativeReply('6C421742129F9');
Trace("HCIPINCodeRequestNegativeReply returned: ", result,
"\n");
```

HCIPINCodeRequestReply()

HCIPINCodeRequestReply(Address, PINCode)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device for which PIN code will be used.		
PINCode	PIN code to use when connecting to the device.		Must be 1 to 16 characters in length.

Return value

- "Success"
- "Failure"

Comments

Specifies the PIN code to use for a connection.

Example

```
result = HCIPINCodeRequestReply('6C421742129F9', "New PIN
Code");
Trace("HCIPINCodeRequestReply returned: ", result, "\n");
```

HCIQoSSetup()

HCIQoSSetup(Address, ServiceType, TokenRate,
PeakBandwidth, Latency, DelayVariation)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of the remote		
ServiceType	The one-byte service type: 0=No traffic; 1=Best effort; 2=Guaranteed		
TokenRate	The four-byte token rate value in bytes per second		

Parameter	Meaning	Default Value	Comments
Peak Bandwidth	The four-byte peak band-width value in bytes per sec-ond		
Latency	The four-byte latency value in microsec-onds		
Delay Variation	The four-byte delay varia-tion value in microseconds		

Return values

Returns a list with eight values: *status, HCI handle, flags, service type, token rate, peak bandwidth, latency,* and *delay variation.*

Status (element 0) is one of the following:

- "Success"
- "Failure"

HCI handle (element 1) is the handle for the ACL connection.

Flags (element 2) is a one-byte value reserved for future use.

Service type (element 3) is the one-byte service type. (0=No traffic; 1=Best effort; 2=Guaranteed.)

Token rate (element 4) is the four-byte token rate value in bytes per second.

Peak bandwidth (element 5) is the four-byte peak bandwidth value in bytes per second.

Latency (element 6) is the four-byte latency value in microseconds.

Delay variation (element 7) is the four-byte delay variation value in micro-seconds.

Comments

This command is used to specify Quality of Service parameters for the con-nection.

Example

```
QoS = HCIQoSSetup('00803713BDF0', 2, 0, 0, 0x12345678,
0x23456789);
```

```
Trace("Merlin's Wand Link QoS Setup returned: ", QoS[0],
"\n");
if (QoS[0] == "Success")
{
   Trace(" Connection Handle : 0x", QoS[1], "\n");
   Trace(" Flags             : 0x", QoS[2], "\n");
   Trace(" Service Type      : 0x", QoS[3], "\n");
   Trace(" Token Rate        : 0x", QoS[4], "\n");
   Trace(" Peak Bandwidth    : 0x", QoS[5], "\n");
   Trace(" Latency           : 0x", QoS[6], "\n");
   Trace(" Delay Variation   : 0x", QoS[7], "\n\n");
}
```

HCIReadAuthenticationEnable()

`HCIReadAuthenticationEnable()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *authentication enable*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Authentication enable (element 1) is the one-byte authentication enable value. (0=Authentication disabled; 1=Authentication enabled.)

Comments

This command will read the value for AuthenticationEnable parameter.

Example

```
result = HCIReadAuthenticationEnable();
if(result[0] == "Success")
Trace("Merlin's Wand Authentication Enabled : ", result[1],
"\n");
```

HCIReadBDADDR()

`HCIReadBDADDR()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *address*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Address (element 1) is the address of the local device.

Comments

This command is used to read the value for the BD_ADDR parameter. The BD_ADDR is a 48-bit unique identifier for a Bluetooth device.

Example

```
LocalAddress = HCIReadBDADDR();
if(LocalAddress [0] == "Success")
Trace("Local BDADDR:", LocalAddress[1], "\n");
```

HCIReadBufferSize()

`HCIReadBufferSize()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with five values: *status, ACL packet length, SCO packet length, ACL number of packets,* and *SCO number of packets.*

Status (element 0) is one of the following:

- "Success"
- "Failure"

ACL packet length (element 1) is the two-byte value of the maximum length (in bytes) of the data portion of each HCI ACL data packet that the Host Controller is able to accept.

SCO packet length (element 2) is the one-byte value of the maximum length (in bytes) of the data portion of each HCI SCO data packet that the Host Controller is able to accept.

ACL number of packets (element 3) is the total number of HCI ACL data packets that can be stored in the data buffers of the Host Controller.

SCO number of packets (element 4) is the total number of HCI SCO data packets that can be stored in the data buffers of the Host Controller.

Comments

This command is used to read the maximum size of the data portion of SCO and ACL data packets sent from the Host to Host Controller.

Example

```
Trace("Local Buffer parameters\n");
result = HCIReadBufferSize();
Trace(" HCIReadBufferSize() returned: ",   result[0],
"\n");
if (result[0] == "Success")
{
   Trace(" ACL Data Packet Length     : 0x", result[1],
"\n");
   Trace(" SCO Data Packet Length     : 0x", result[2],
"\n");
   Trace(" Total Num ACL Data Packets : 0x", result[3],
"\n");
   Trace(" Total Num SCO Data Packets : 0x", result[4],
"\n");
}
```

HCIReadClockOffset()

`HCIReadClockOffset(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to connect with.		

Return values

Returns a list with two values: *status* and *offset*.

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Offset (element 1) is the two-byte value of the clock offset.

Comments

Reads the clock offset to remote devices.

Example

```
result = HCIReadClockOffset();
Trace("HCIReadClockOffset returned: ", result[0], "\n");
if (result[0] == "Success")
{
   Trace("Clock offset is: 0x", result[1], "\n");
}
```

HCIReadConnectionAcceptTimeout()

`HCIReadConnectionAcceptTimeout()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *timeout*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Timeout (element 1) is the two-byte value of the timeout, interpreted as multiples of 0.625-msec intervals.

Comments

Reads the current timeout interval for connection. The timeout value defines the time duration from when the Host Controller sends a Connection Request event until the Host Controller automatically rejects an incoming connection.

Example

```
result = HCIReadConnectionAcceptTimeout();
Trace("ReadConnectionAcceptTimeout returned: ", result[0],
"\n");
if (result[0] == "Success")
{
   Trace("Timeout value is: 0x", result[1], "\n");
}
```

HCIReadCountryCode()

`HCIReadCountryCode()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *country code*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Country code (element 1) is the one-byte country code value. (0=North America and Europe; 1=France.)

Comments

Reads the country code value. This value defines which range of frequency band of the ISM 2.4-GHz band is used by the device.

Example

```
result = HCIReadCountryCode();
Trace("HCIReadCountryCode returned: ", result[0], "\n");
if (result[0] == "Success")
{
   Trace("Country code is: 0x", result[1], "\n");
}
```

HCIReadCurrentIACLAP()

`HCIReadCurrentIACLAP()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

Returns a list with two values: *status* and *Current IAC LAP*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Current IAC LAP (element 1) is the 3-byte value of the LAPs (Lower Address Part) that make up the current IAC (Inquiry Access Code).

Comments

Reads the number and values of the currently used IAC LAPs.

Example

```
result = HCIReadCurrentIACLAP();
if(result[0] == "Success")
{
   Trace("Current number of used IAC LAPs is: ", result[1],
"\n");
   if(result[1] > 0)
   {
      Trace("Currently used IAC LAPs are:");
      for(i = 0; i < result[1]; i = i + 1)
      {
         Trace(" 0x", result[2 + i]);
      }
      Trace("\n\n");
   }
}
```

HCIReadEncryptionMode()

`HCIReadEncryptionMode()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *encryption mode*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Encryption mode (element 1) is the one-byte encryption mode value. (0=Encryption disabled; 1=Encryption enabled for point-to-point packets only; 2=Encryption enabled for both point-to-point and broadcast packets.)

Comments

Reads the encryption mode value. This value controls whether the local device requires encryption to the remote device at connection setup.

Example

```
result = HCIReadEncryptionMode();
Trace("HCIReadEncryptionMode returned: ", result[0], "\n");
if (result[0] == "Success")
```

```
{
   Trace("Encryption mode is: 0x", result[1], "\n");
}
```

HCIReadLinkPolicySettings()

HCIReadLinkPolicySettings(Address)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		

Return value

Returns the following list of values: *status* and *link policy settings.*

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Link policy settings (element 1) is the two-byte value of the link policy settings.

Comments

Reads the value of the Link_Policy_Settings parameter for the device.

Example

```
Device = '010203040506';
result = HCIReadLinkPolicySettings(Device);
Trace("HCIReadLinkPolicySettings returned: ", result[0],
"\n");
if (result[0] == "Success")
{
   Trace("Link Policy Settings : ", result[1] ,"\n");
}
```

HCIReadLinkSupervisionTimeout()

`HCIReadLinkSupervisionTimeout(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		

Return value

Returns the following list of values: *status* and *link supervision timeout.*

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Link supervision timeout (element 1) is the timeout, interpreted as multiples of 0.625-msec intervals.

Comments

Reads the value for the Link_Supervision_Timeout parameter for the device.

Example

```
Device = '010203040506';
result = HCIReadLinkSupervisionTimeout(Device);
Trace("HCIReadLinkSupervisionTimeout returned: ",
result[0], "\n");
if (result[0] == "Success")
{
   Trace("Link Supervision Timeout is: ", result[1] ,"\n");
}
```

HCIReadLocalName()

`HCIReadLocalName()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *name*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Name (element 1) is a string representing the device name.

Comments

Reads the "user-friendly" name of the local Bluetooth device.

Example

```
result = HCIReadLocalName();
Trace("HCIReadLocalName returned: ", result[0], "\n");
if (result[0] == "Success")
{
   Trace("Local device name is: ", result[1], "\n");
}
```

HCIReadLocalSupportedFeatures()

`HCIReadLocalSupportedFeatures()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *features*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Features (element 1) is the eight-byte bit mask list of Link Manager Protocol features.

Comments

Reads the LMP supported features for the local device.

Example

```
result = HCIReadLocalSupportedFeatures();
Trace("HCIReadLocalSupportedFeatures returned: ",
result[0], "\n");
if (result[0] == "Success")
{
   Trace("Local supported features data is: ", result[1],
"\n");
}
```

HCIReadLocalVersionInformation()

`HCIReadLocalVersionInformation()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with six values: *status, HCI version, HCI revision, LMP version, manufacturer name,* and *LMP subversion.*

Status (element 0) is one of the following:

- "Success"
- "Failure"

HCI version (element 1) is the one-byte HCI version value. (0=1.0B, 1=1.1.)

HCI revision (element 2) is the two-byte HCI revision value.

LMP version (element 3) is the one-byte Link Manager Protocol version value.

Manufacturer name (element 4) is the two-byte manufacturer name of the Bluetooth hardware.

LMP subversion (element 5) is the two-byte Link Manager Protocol subversion value.

Comments

Reads the version information for the local device.

Example

```
result = HCIReadLocalVersionInformation();
Trace("HCIReadLocalVersionInformation returned: ",
result[0], "\n");
if (result[0] == "Success")
{
   Trace("HCI version is: 0x",       result[1], "\n");
   Trace("HCI revision is: 0x",      result[2], "\n");
   Trace("LMP version is: 0x",       result[3], "\n");
   Trace("Manufacturer name is: 0x", result[4], "\n");
```

```
      Trace("LMP subversion is: 0x",    result[5], "\n");
   }
```

HCIReadLoopbackMode()

HCIReadLoopbackMode()

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *loopback mode*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Loopback mode (element 1) is the one-byte loopback mode value. (0=No loopback mode; 1=Local loopback mode; 2=Remote loopback mode.)

Comments

Reads the loopback mode value. This value determines the path by which the Host Controller returns information to the Host.

Example

```
   result = HCIReadLoopbackMode();
   Trace("HCIReadLoopbackMode returned: ", result[0], "\n");
   if (result[0] == "Success")
   {
      Trace("Loopback mode is: 0x", result[1], "\n");
   }
```

HCIReadNumberOfSupportedIAC()

HCIReadNumberOfSupportedIAC()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

Returns a list with two values: *status* and *number of supported IAC*.

Status is one of the following:

- "Success"

- "Failure"

Number of supported IAC is a 1-byte value that specifies the number of Inquiry Access Codes that the local Bluetooth device can listen for at one time.

Comments

Reads the number of supported IACs.

Example

```
result = HCIReadNumberOfSupportedIAC ();
Trace("HCIReadNumberOfSupportedIAC returned: ", result[0],
"\n");
if (result[0] == "Success")
{
   Trace("The number of supported IAC is: ", result[1],
"\n\n");
}
```

HCIReadPageScanMode()

`HCIReadPageScanMode()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *page scan mode*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Page scan mode (element 1) is the one-byte page scan mode value. (0=Mandatory page scan mode; 1=Optional page scan mode I; 2=Optional page scan mode II; 3=Optional page scan mode III.)

Comments

Reads the page scan mode value. This value indicates the mode used for default page scan.

Example

```
result = HCIReadPageScanMode();
Trace("HCIReadPageScanMode returned: ", result[0], "\n");
if (result[0] == "Success")
{
        Trace("Page scan mode is: 0x", result[1], "\n");
    }
```

HCIReadPageScanPeriodMode()

HCIReadPageScanPeriodMode()

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *page scan period mode*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Page scan period mode (element 1) is the one-byte page scan period mode value. (0=P0; 1=P1; 2=P2.)

Comments

Reads the page scan period mode value. Each time an inquiry response message is sent, the Bluetooth device will start a timer, the value of which depends on the page scan period mode.

Example

```
    result = HCIReadPageScanPeriodMode();
    Trace("HCIReadPageScanPeriodMode returned: ", result[0],
    "\n");
    if (result[0] == "Success")
    {
        Trace("Page scan period mode is: 0x", result[1], "\n");
    }
```

HCIReadPageTimeout()

HCIReadPageTimeout()

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *page timeout*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Page timeout (element 1) is the two-byte page timeout value, in increments of 0.625-msec intervals.

Comments

Reads the page timeout value. This value defines the maximum time the local Link Manager will wait for a baseband page response from the remote device at a locally initiated connection attempt.

Example

```
result = HCIReadPageTimeout();
Trace("HCIReadPageTimeout returned: ", result[0], "\n");
if (result[0] == "Success")
{
   Trace("Page timeout is: 0x", result[1], "\n");
}
```

HCIReadPINType()

HCIReadPINType()

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *PIN type*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

PIN type (element 1) is the one-byte PIN type. (0=Variable PIN; 1=Fixed PIN.)

Comments

Reads the PIN type, which determines whether the Host supports variable PIN codes or only a fixed PIN code.

Example

```
result = HCIReadPINType();
Trace("HCIReadPINType returned: ", result[0], "\n");
if (result[0] == "Success")
{
   Trace("PIN type is: 0x", result[1], "\n");
}
```

HCIReadRemoteSupportedFeatures()

`HCIReadRemoteSupportedFeatures(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to connect with.		

Return values

Returns a list with two values: *status* and *features*.

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Features (element 1) is the eight-byte bit mask list of Link Manager Protocol features.

Comments

Reads the LMP supported features for the specified device. An ACL connection with the device is required.

Example

```
Device = '010203040506';
result = HCIReadRemoteSupportedFeatures(Device);
Trace("HCIReadRemoteSupportedFeatures returned: ",
result[0], "\n");
if (result[0] == "Success")
{
   Trace("Remote supported features data is: 0x",
result[1], "\n");
}
```

HCIReadRemoteVersionInformation()

`HCIReadRemoteVersionInformation(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to connect with.		

Return values

Returns a list with four values: *status, LMP version, manufacturer name,* and *LMP subversion.*

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

LMP version (element 1) is the one-byte Link Manager Protocol version value.

Manufacturer name (element 2) is the two-byte manufacturer name of the Bluetooth hardware.

LMP subversion (element 3) is the two-byte Link Manager Protocol subversion value.

Comments

Reads the version information for the specified device. An ACL connection with the device is required.

Example

```
Address = '010203040506';
result = HCIReadRemoteVersionInformation(Address);
Trace("HCIReadRemoteVersionInformation returned: ",
result[0], "\n");
if (result[0] == "Success")
{
   Trace("LMP version is: 0x", result[1], "\n");
   Trace("Manufacturer name is: 0x", result[2], "\n");
   Trace("LMP subversion is: 0x", result[3], "\n");
}
```

HCIReadScanEnable()

`HCIReadScanEnable()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "GENERAL_ACCESSIBLE"
- "LIMITED_ACCESSIBLE"
- "NOT_ACCESSIBLE"
- "CONNECTABLE_ONLY"
- "Failure"

Comments

Retrieves the current accessible mode of Merlin's Wand.

Example

```
Trace("Merlin's Wand accessible mode: ",
HCIReadScanEnable());
```

HCIReadSCOFlowControlEnable()

`HCIReadSCOFlowControlEnable()`

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *SCO flow control enable*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

SCO flow control enable (element 1) is the one-byte SCO flow control value. (0=SCO flow control disabled; 1=SCO flow control enabled.)

Comments

Reads the SCO flow control enable value. This value determines whether the Host Controller will send Number Of Completed Packets events for SCO Connection Handles.

Example

```
result = HCIReadSCOFlowControlEnable();
Trace("HCIReadSCOFlowControlEnable returned: ", result[0],
"\n");
if (result[0] == "Success")
{
   Trace("SCO flow control enable is: 0x", result[1],
"\n");
}
```

HCIReadStoredLinkKey()

HCIReadStoredLinkKey(Address, ReadAll)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device that will have its link key read		
ReadAll	Boolean value that indicates whether to read only the specified address's link key, or all link keys	0	0 or 1

Return values

Returns a list with two values: *status* and *data*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Data (element 1) is a list containing zero or more pairs of the following two values:

- BDADDR: the Bluetooth Address that the link key corresponds to
- LinkKey: the link key for the specified address

Comments

Attempts to read the stored link key for the specified address or for all addresses, depending on the value of ReadAll.

Example

```
result = HCIReadStoredLinkKey('6E8110AC0008', 1);
Trace("HCIReadStoredLinkKey() returned: ", result[0],
"\n\n");

if (result[0] == "Success")
{
   list = result[1];
   i = 0;
   while (list[(i*2)] != null)
   {
      Trace("***********************\n");
      Trace("BDADDR: ", list[(i*2)], "\n");
      Trace("Link Key: ", list[(i*2)+1], "\n");
      Trace("***********************\n");
      i = i + 1;
   }
}
```

HCIReadVoiceSetting()

HCIReadVoiceSetting()

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with two values: *status* and *voice setting*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

Voice setting (element 1) is the 10-bit voice setting value.

Comments

Reads the voice setting value. This value controls all settings for voice connections.

Example

```
result = HCIReadVoiceSetting();
Trace("HCIReadVoiceSetting returned: ", result[0], "\n");
if (result[0] == "Success")
{
   Trace("Voice setting is: 0x", result[1], "\n");
}
```

HCIRejectConnectionRequest()

`HCIRejectConnectionRequest()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"

Comments

Sets the accept connection request variable to False.

Example

```
status = HCIRejectConnectionRequest();
Trace("HCIRejectConnectionRequest returned: ", status,
"\n\n");
```

HCIRemoveSCOConnection()

`HCIRemoveSCOConnection(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to connect with		

Return value

- "Success"
- "Not connected"
- "Failure"

Comments

Removes an existing SCO connection associated with the specified device.

Example

```
result = HCIRemoveSCOConnection(Devices[0]);
```

HCIReset()

`HCIReset()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"
- "Invalid parameter"
- "Failed: Invalid Type"
- "Failed: HCI initialization error"

Comments

Resets the Host Controller and Link Manager.

Example

```
result = HCIReset();
```

HCIRoleDiscovery()

`HCIRoleDiscovery(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device relative to which we want to know our role		A connection must exist with this address for Role Discovery to work.

Return value

- "Master"
- "Slave"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Comments

Attempts to discover the role of our device relative to the specified device.

Example

```
result = HCIRoleDiscovery('6E8110AC0108');
```

```
if(result != "Success")
{
   MessageBox(result, "Failed to get role!");
}
else
{
   Trace("Our role is: ", result, "\n\n");
}
```

HCISetConnectionEncryption()

`HCISetConnectionEncryption(Address, SetEncryption)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device whose encryption to enable or disable		
SetEncryption	Boolean value that indicates whether to enable or disable encryption	0	0 or 1 A connection must be established and authenticated before you can enable encryption successfully

Return value

- "Success"
- "Failure"
- "Timed Out"
- "Failed: Device not found"
- "Not connected"

Comments

Enables and disables the link-level encryption for the address specified

Example

```
result = HCISetConnectionEncryption('6E8110AC0108', 0);
if(result != "Success")
{
   MessageBox(result, "Failed to disable encryption!");
}
```

HCISetEventFilter()

HCISetEventFilter(FilterType, FilterConditionType, Condition)

Parameter	Meaning	Default Value	Comments
FilterType	Filter type: 0 = Clear all filters; 1 = Inquiry result; 2 = Connection setup; 3-255 = Reserved		If value 0 is used, no other parameters should be supplied.
Filter Condition Type	Type of filter condition.		This parameter has different meanings depending on the filter type.
Condition	Details of the filter to be set.		Must be entered as a series of bytes within brackets, e.g., [0x1, 0x12, 0x0F]. Byte values must be entered in hex notation separated by commas.

Return value

- "Success"
- "Failure"
- "Invalid parameter"

Comments

Instructs the Host Controller to send only certain types of events to the Host.

Examples

```
# Clear All Filters
result = HCISetEventFilter(0);
Trace("Result of clearing all filters: ", result, "\n");

# Inquiry Result
result = HCISetEventFilter(1, 2,
[0xA,0x1,0x24,0x12,0xFB,0xAA]);
Trace("Result of Inquiry Result filter: ", result, "\n");

# Connection Setup
result = HCISetEventFilter(2, 0, [0x1]);
Trace("Result of Connection Setup filter: ", result, "\n");
```

HCISniffMode()

`HCISniffMode(Address, MaxInterval, MinInterval, Attempt, Timeout)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		
MaxInterval	Maximum number of 0.625-msec intervals between sniff periods.		Range is 0x0001 to 0xFFFF (0.625 msec to 40.9 sec).
MinInterval	Minimum number of 0.625-msec intervals between sniff periods.		Range is 0x0001 to 0xFFFF (0.625 msec to 40.9 sec).
Attempt	Number of receive slots for sniff attempt.		Range is 0x0001 to 0x7FFF (0.625 msec to 20.5 sec).
Timeout	Number of receive slots for sniff time-out.		Range is 0x0001 to 0x7FFF (0.625 msec to 20.5 sec).

Return value

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Comments

Enters Sniff mode with parameters as specified.

Example

```
Device = '010203040506';
result = HCISniffMode(Device, 0xFFFF, 100, 0x3FF6, 0x7FFF);
Trace("HCISniffMode result is: ", result, "\n");
```

HCISwitchRole()

`HCISwitchRole(Address, Role)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		
Role			Values: "Master", "Slave"

Return value

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"
- "Invalid parameter"

Comments

Switches the current role of the device in the piconet.

Example

```
Device = '010203040506';
result = HCISwitchRole(Device, "Slave");
Trace("HCISwitchRole result is: ", result, "\n\n");
```

HCIWriteAuthenticationEnable()

`HCIWriteAuthenticationEnable(AuthenticationEnable)`

Parameter	Meaning	Default Value	Comments
AuthenticationEnable	Authentication value: 0 = Authentication disabled; 1 = Authentication enabled for all connections; 2-255 = Reserved	0	

Return value

- "Success"

- "Failure"
- "Invalid parameter"

Comments

Controls whether the local device is required to authenticate the remote device at connection setup.

Example

```
result = HCIWriteAuthenticationEnable(0);
```

HCIWriteConnectionAcceptTimeout()

`HCIWriteConnectionAcceptTimeout(Interval)`

Parameter	Meaning	Default Value	Comments
Interval	Number of 0.625-msec intervals before the connection request times out.	0x1FA0 (= 5 sec)	Range is 0x0001 to 0xB540 (0.625 msec to 29 sec).

Return value

- "Success"
- "Failure"
- "Invalid parameter"

Comments

Sets a timeout interval for connection. The parameter defines the time duration from when the Host Controller sends a Connection Request event until the Host Controller automatically rejects an incoming connection.

Example

```
result = HCIWriteConnectionAcceptTimeout(0x1234);
```

HCIWriteCurrentIACLAP()

`HCIWriteCurrentIACLAP(NumCurrentIACs, IACLAPs)`

Parameter	Meaning	Default Value	Comments
NumCurrent IACs	Number of current IACs.		Must be 1 or 2.

Parameter	Meaning	Default Value	Comments
IACLAPs	List of IAC_LAPs, each with a value in the range 0x9E8B00-0x9E8B3F.		The number of values in this list must match the NumCurrentIACs parameter.

Return value

- "Success"
- "Failure"
- "Invalid parameter"

Comments

Writes the number and values of the IAC LAPs to be used. One of the values has to be the General Inquiry Access Code, 0x9E8B33.

Example

```
result = HCIWriteCurrentIACLAP(2, 0x9E8B33, 0x9E8B34);
Trace("Result of HCIWriteCurrentIACLAP: ", result, "\n\n");
```

HCIWriteEncryptionMode()

HCIWriteEncryptionMode(EncryptionMode)

Parameter	Meaning	Default Value	Comments
Encryption Mode	Encryption mode: 0 = Encryption disabled; 1 = Encryption enabled for point-to-point packets only; 2 = Encryption enabled for both point-to-point and broadcast packets; 3-255=Reserved	0	

Return value

- "Success"

- "Failure"
- "Invalid parameter"

Comments

Controls whether the local device requires encryption to the remote device at connection setup.

Example

```
result = HCIWriteEncryptionMode(0);
```

HCIWriteLinkPolicySettings()

`HCIWriteLinkPolicySettings(Address, LinkPolicySettings)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		
LinkPolicy Settings			Range is 0x0000-0x8000.

Return value

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Comments

Writes the value for the Link_Policy_Settings parameter for the device.

Example

```
Device = '010203040506';
result = HCIWriteLinkPolicySettings(Device, 0xF);
Trace("HCIWriteLinkPolicySettings result is: ", result,
"\n\n");
```

HCIWriteLinkSupervisionTimeout()

`HCIWriteLinkSupervisionTimeout(Address, Timeout)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device in question		
Timeout	Number of 0.625-msec intervals before connection request times out.		Range is 0x0001 to 0xFFFF (0.625 msec to 40.9 sec).

Return value

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not connected"

Comments

Writes the value for the Link_Supervision_Timeout parameter for the device.

Example

```
Device = '010203040506';
result = HCIWriteLinkSupervisionTimeout(Device, 0x7D00);
Trace("HCIWriteLinkSupervisionTimeout result is: ",
result[0], "\n\n");
```

HCIWriteLoopbackMode()

`HCIWriteLoopbackMode(LoopbackMode)`

Parameter	Meaning	Default Value	Comments
Loopback Mode	Loopback mode: 0 = No loop-back mode; 1 = Local loopback mode; 2 = Remote loopback mode; 3-255 = Reserved	0	

Return value

- "Success"
- "Failure"
- "Invalid parameter"

Comments

Determines the path by which the Host Controller returns information to the Host.

Example

```
result = HCIWriteLoopbackMode(2);
```

HCIWritePageTimeout()

`HCIWritePageTimeout(Interval)`

Parameter	Meaning	Default Value	Comments
Interval	Number of 0.625-msec intervals before the connection attempt times out.	0x2000 (= 5.12 sec)	Range is 0x0001 to 0xFFFF (0.625 msec to 40.9 sec).

Return value

- "Success"
- "Failure"

- "Invalid parameter"

Comments

Sets the maximum time the local Link Manager will wait for a baseband page response from the remote device at a locally initiated connection attempt.

Example

```
result = HCIWritePageTimeout(0x4000);
```

HCIWritePINType()

`HCIWritePINType(PINType)`

Parameter	Meaning	Default Value	Comments
PINType	PIN type: 0=Variable PIN; 1=Fixed PIN		Range is 0 to 1.

Return value

- "Success"
- "Failure"
- "Invalid parameter"

Comments

Determines whether the Host supports variable PIN codes or only a fixed PIN code.

Example

```
result = HCIWritePINType(0);
```

HCIWriteScanEnable()

`HCIWriteScanEnable(AccessibleMode)`

Parameter	Meaning	Default Value	Comments
Accessible Mode	Access mode to set Merlin's Wand	"GENERAL_ ACCESSI- BLE"	Mode can be one of: "GENERAL_ACCESSIBLE", "NOT_ACCESSIBLE", "CONNECTABLE_ONLY"

Return value

- "Success"

- "Timed out"
- "Failure"

Comments

Sets the accessible mode of Merlin's Wand.

Example

```
HCIWriteScanEnable("CONNECTABLE_ONLY");
```

HCIWriteStoredLinkKey()

HCIWriteStoredLinkKey(Address, LinkKey)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device that will have its link key stored		
LinkKey	String containing the link key to be stored		Up to 32 Hex digits

Return value

- "Success"
- "Failure"

Comments

Attempts to store the link key for the specified address. If a link key already exists for the specified address, it will be overwritten.

Example

```
result = HCIWriteStoredLinkKey('6E8110AC0108', "ABC123");
Trace("HCIWriteStoredLinkKey() returned: ", result,
"\n\n");
```

HCIWriteVoiceSettings()

`HCIWriteVoiceSettings(Address, VoiceSetting)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device whose voice settings to write		
VoiceSetting	Three-digit hex value containing the voice settings		Possible values: 0x0060=CVSD coding 0x0061=u-Law coding 0x0062=A-law coding

Return value

- "Success"
- "Failure"
- "Timed Out"
- "Failed: Device not found"
- "Not connected"

Comments

Attempts to write the voice settings for the specified address. A connection must be established before voice settings can be written.

Example

```
result = HCIWriteVoiceSettings('6E8110AC0108', 0x0060);
Trace("HCIWriteVoiceSettings() returned: ", result,
"\n\n");
```

C.5 OBEX Commands

OBEXClientAbort()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "ClientAbort Complete"
- "ClientAbort Error"

Comments

Aborts a ClientGet or ClientPut operation.

Example

OBEXClientConnect()

`OBEXClientConnect(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to connect with		

Return value

- "Success"
- "Failure"
- "Failed: Busy"
- "Failed: Not connected"
- "Failed: Packet too small"

Comments

Establishes an OBEX client connection with the specified device.

Example

```
result = OBEXClientConnect(Devices[0]);
if(result != "Success")
{
   MessageBox("Failed to establish OBEX connection.");
}
```

OBEXClientDeinit()

`OBEXClientDeinit()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Failure"

Comments

This command is obsolete. It is provided for backward compatibility only. (The application is initialized as an OBEX client at startup and cannot be deinitialized.)

Example

```
result = OBEXClientDeinit();
```

OBEXClientDisconnect()

`OBEXClientDisconnect()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"
- "Failed: Busy"
- "Failed: Not connected"
- "Failed: Packet too small"

Comments

Breaks the current OBEX client connection.

Example

```
result = OBEXClientDisconnect();
```

OBEXClientGet()

`OBEXClientGet(RemotePath, LocalPath)`

Parameter	Meaning	Default Value	Comments
RemotePath	Path and name of object to be retrieved from server.		Path is relative to server's OBEX directory. Example: If the OBEX directory is C:\temp, a RemotePath of "file.txt" would cause the client to retrieve "C:\temp\file.txt"

Parameter	Meaning	Default Value	Comments
LocalPath	Path and name of object to be created on client.	RemotePath argument	If omitted, object will be stored to the local OBEX directory with the name it has on the server. If specified as a relative path (i.e., without a drive letter), the path will be considered relative to the OBEX directory. If specified as a full path (i.e., with a drive letter), the object will be stored to the exact name and path specified.

Return value

- "Success"
- "Failure"
- "Failed: Busy"
- "Failed: Not connected"
- "Failed: Packet too small"
- "Failed: Invalid handle"

Comments

Retrieves object from a server and saves it to the client.

If directory names are included in either path argument, **be sure to use double-slashes to separate components** (e.g., "temp1\\temp2\\file-name.txt"). Using single slashes will cause errors.

Note that the second argument may be omitted, in which case the object will be stored to the client's OBEX directory with the same name it has on the server.

Examples

In these examples, the local OBEX directory is assumed to be c:\obexdir.

```
#store file to "file.txt" in local OBEX directory
#  (i.e., c:\obexdir\file.txt)
OBEXClientGet("file.txt");

#store file to "newfile.txt" in temp dir under OBEX dir
#  (i.e., c:\obexdir\temp\newfile.txt)
OBEXClientGet("file.txt", "temp\\newfile.txt");

#store file to "file.txt" in C:\temp
OBEXClientGet("file.txt", "C:\\temp\\file.txt");

#get file from a directory below the server's OBEX dir,
#  and save it with the same name to the same directory
# below the local OBEX dir (i.e.,
"c:\obexdir\temp\file.txt")
OBEXClientGet("temp\\file.txt");
```

OBEXClientInit()

`OBEXClientInit()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"

Comments

This command is obsolete. It is provided for backward compatibility only. (The application is initialized as an OBEX client at startup and cannot be deinitialized.)

Example

```
result = OBEXClientInit();
```

OBEXClientPut()

`OBEXClientPut(LocalPath, RemotePath)`

Parameter	Meaning	Default Value	Comments
LocalPath	Full (*not* relative) path and name of file to be sent from client.		
RemotePath	Path and name of object to be stored on server.	Name-only portion of LocalPath argument	Path is relative to server's OBEX directory. Example: If the server's OBEX directory is C:\Temp, a RemotePath of "file.txt" would cause the server to save the file to "C:\Temp\file.txt". Note that you cannot save a file to an absolute path on the server.

Return value

- "Success"
- "Failure"
- "Failed: Busy"
- "Failed: Invalid handle"

- "Failed: Invalid parameter"
- "Failed: Media busy"
- "Failed: Not connected"
- "Failed: Packet too small"

Comments

Sends a file to the OBEX directory of the server.

If directory names are included in either path argument, **be sure to use double-slashes to separate components** (e.g., "temp1\\temp2\\file-name.txt"). Using single slashes will cause errors.

Note that the second argument may be omitted, in which case the object will be stored to the server's OBEX directory with the same name it has on the client.

Examples

In these examples, the server's OBEX directory is assumed to be c:\obexdir.

```
#store file to "file.txt" in server's OBEX directory
#  (i.e., c:\obexdir\file.txt)
OBEXClientPut("c:\\temp\\file.txt");

#store file to "newfile.txt" in server's OBEX dir
#  (i.e., c:\obexdir\newfile.txt)
OBEXClientPut("c:\\temp\\file.txt", "newfile.txt");

#store file to "newfile.txt" in temp dir under OBEX dir
#  (i.e., c:\obexdir\temp\newfile.txt)
OBEXClientPut("c:\\temp\\file.txt", "temp\\newfile.txt");
```

OBEXClientSetPath()

`OBEXClientSetPath(Path, Flags)`

Parameter	Meaning	Default Value	Comments
Path	New path to set		Path is relative to server's current working directory. Cannot begin "C:" or "\\\\".

Parameter	Meaning	Default Value	Comments
Flags	SetPath flags: 0=No flags 1=Back up one level 2=Don't create specified folder if it doesn't exist 3=Back up one level and don't create specified folder		When backup is set (flag = 1 or 3), the working directory is backed up one level before the specified directory is appended (e.g., if the server's current working directory is C:\Temp, a SetPath of "Temp2" with a flag of 1 would change the directory to C:\Temp2). To set path to the OBEX root directory, use an empty path and a flag of 0 or 2.

Return value

- "Success"
- "Failure"
- "Failed: Busy"
- "Failed: Not connected"
- "Failed: Packet too small"
- "Failed: Invalid parameter"

Comments

Temporarily changes a server's current working directory, accessed by clients during `ClientGet` and `ClientPut` operations. The device must be connected to an OBEX server before the command can be successfully executed. The change is lost when the connection is broken. Note that the server's OBEX root directory cannot be changed with this command.

If the path includes multiple levels, **be sure to use double-slashes to separate components** (e.g., "temp1\\temp2"). Using single slashes will cause errors.

Example

```
#set path to <root>
status = OBEXClientSetPath("", 0);
Sleep(1000);

#set path to <root>\temp2
status = OBEXClientSetPath("temp2", 0);
Sleep(1000);

#set path to <root>\temp2\temp3
status = OBEXClientSetPath("temp3", 0);
Sleep(1000);

#set path to <root>\temp2
```

```
status = OBEXClientSetPath("", 1);
Sleep(1000);

#keep path at <root>\temp2 (assuming <root>\temp2\temp4
doesn't exist)
status = OBEXClientSetPath("temp4", 2);
Sleep(1000);

#set path to <root>\temp3\temp4
status = OBEXClientSetPath("temp3\\temp4", 1);
```

OBEXServerDeinit()

`OBEXServerDeinit()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"
- "Failed: Busy"

Comments

Deinitializes an OBEX server.

Example

```
result = OBEXServerDeinit();
```

OBEXServerSetPath(Path)

`OBEXServerSetPath(Path)`

Parameter	Meaning	Default Value	Comments
Path	Path to be used as the OBEX root directory on the server		Path must be fully specified (e.g., "C:\\temp" rather than "temp")

Return value

- "Success"
- "Failure"
- "Failed: Device must be initialized as a server"

Comments

Sets the OBEX root directory on a server. This path is accessed by clients during remote `ClientGet` and `ClientPut` operations. The device must be initialized as a server before the command can be successfully executed.

In the path, **be sure to use double-slashes to separate components** (e.g., "C:\\temp\\temp2"). Using single slashes will cause errors.

Example

```
status = OBEXServerInit();
if ( status == "Success" )
{
   status = OBEXServerSetPath("c:\\temp");
}
Trace("OBEXServerSetPath returned: ", status, "\n\n");
```

C.6 RFCOMM Commands

`RFCloseClientChannel()`

`RFCloseClientChannel(Address, DLCI)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()

Return value

- "Success"
- "Not connected"
- "Failure"
- "Timed out"

Comments

Closes an RFCOMM channel

Example

```
RFCloseClientChannel(Devices[0], DLCI);
```

RFOpenClientChannel()

`RFOpenClientChannel(Address, ServerID)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		
ServerID	Service ID for RFCOMM channel		

Return value

The return value from RFOpenClientChannel is a list containing up to two elements. The first element is the status of the command and is one of the following strings:

- "Success"
- "Failure"
- "Timed out"
- "Not connected"
- "Restricted"

If the return value is "Success", the second element in the list is the DLCI of the established connection.

Comments

An ACL connection must already be established with the device.

Example

```
result = RFOpenClientChannel(Devices[0], 1);
if(result[0] == "Success")
{
   Trace("Successfully connected with DLCI ", result[1],
"\n");
   # Send some data over RFCOMM
}
```

RFRegisterServerChannel()

`RFRegisterServerChannel()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- Server channel ID
- "Failure"

Comments

Example

```
channel = RFRegisterServerChannel();
if(channel != "Failure")
{
   Trace("Channel ID is ", channel);
}
```

RFSendData()

`RFSendData(Address, DLCI, Data)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		Can use "CONNECTED_DEVICE" to send data to a master RFCOMM connection. Note that this will work only if exactly one device is connected via RFCOMM.
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()
Data	Data to send		Data can be a string, 32-bit integer value or a list containing either or both types

Return value

- "Success"
- "Timed out"
- "Not supported" (invalid data type)
- "Not connected"

Comments

An RFCOMM connection must already be established with the device.

Example

```
RFSendData(Devices[0], DLCI, "ATDT 555-1212");
RFSendData("CONNECTED_DEVICE", dlci, "AT+CKPD=200\r\n");
```

RFSendDataFromPipe()

`RFSendDataFromPipe(Address, DLCI, PipeName)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		Can use "CONNECTED_DEVICE" to send data to a master RFCOMM connection
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()
PipeName	Name of the transmit data pipe to get data to send		This pipe must exist.

Return value

- "Success"
- "Timed out"
- "Not supported" (invalid data type)
- "Not connected"
- "Pipe not found"
- "Internal Error"

Comments

An RFCOMM connection must already be established with the device. The pipe specified must already be set up in the Data Transfer Manager. The pipe should not be open when RFSendDataFromPipe is called.

Example

```
RFSendDataFromPipe(Devices[0], dlci, "MyPipe");
RFSendDataFromPipe("CONNECTED_DEVICE", dlci, "Pipe2");
```

RFReceiveData()

`RFReceiveData(Address, DLCI, Timeout)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		Can use "CONNECTED_DEVICE" to receive data from a master RFCOMM connection
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()

Parameter	Meaning	Default Value	Comments
Timeout	Time in ms to wait for an RFCOMM connection	0 (Infinite wait)	Use 0 as the timeout value to wait infinitely

Return value

Returns a list with three values: *status*, *number of bytes*, and *data array.*

Status (element 0) is one of the following:

- "Success"
- "Not connected"
- "Timed out"

Number of bytes (element 1) is the number of bytes received.

Data array (element 2) is the sequence of data bytes received.

Comments

Receives data from a device connected via RFCOMM. Waits Timeout milliseconds (or infinitely if 0 is specified) for the device to begin sending data to Merlin's Wand.

Example

```
#Get the data; stop when no data is received for 5 secs
result = RFReceiveData(Device, DLCI, 5000);
while(result[0] == "Success")
{
   Trace("Number of data bytes received: ", result[1],
"\n");
   result = RFReceiveData(Device, DLCI, 5000);
}
```

RFWaitForConnection()

`RFWaitForConnection(ServerID, Timeout)`

Parameter	Meaning	Default Value	Comments
ServerID	Service ID for RFCOMM channel		
Timeout	Time in ms to wait for an RFCOMM connection	0 (Infinite wait)	Use 0 as the timeout value to wait infinitely.

Return value

Returns a list with three values: *status*, *DLCI*, and *BluetoothDevice*.

Status (element 0) is one of the following:

- "Success"
- "Timed out"
- "Failure"

DLCI (element 1) is the data link connection identifier.

BluetoothDevice (element 2) is the address of the connecting device.

Comments

Waits Timeout milliseconds for a device to establish an RFCOMM connection with Merlin's Wand. This function will block the specified amount of time (or infinitely if 0 is specified) unless a connection is established. If an RFCOMM connection is already present when this function is called, it will immediately return "Success".

Example

```
# Wait 3 seconds for RFCOMM connection
Trace("RFWaitForConnection\n");
result = RFWaitForConnection(1, 3000);
if( result[0] == "Success" )
{
   Trace("Incoming RFCOMM connection DLCI: ", result[1],
"\n");
   Trace("Connecting device address: ", result[2], "\n");
}
```

RFAcceptChannel()

`RFAcceptChannel(bAccept)`

Parameter	Meaning	Default Value	Comments
bAccept	Boolean value indicating whether to accept the channel or not	0	0 or 1

Return value

- "Success"

Comments

Example

```
status = RFAcceptChannel(1);
Trace("RFAcceptChannel returned: ", status, "\n\n");
```

RFAcceptPortSettings()

`RFAcceptPortSettings(bAccept)`

Parameter	Meaning	Default Value	Comments
bAccept	Boolean value indicating whether to accept the channel or not	0	0 or 1

Return value

- "Success"

Comments

Example

```
status = RFAcceptPortSettings(0);
Trace("RFAcceptPortSettings returned: ", status, "\n\n");
```

RFCreditFlowEnabled()

`RFCreditFlowEnabled(Address, DLCI)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		Can use "CONNECTED_DEVICE" to check if credit flow is enabled on a master RFCOMM connection. Note that this will work only if exactly one device is connected via RFCOMM.
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()

Return value

- "Enabled"
- "Disabled"
- "Not Connected"

Comments

Checks to see if credit flow is enabled on a particular RFCOMM connection.

Example

```
result = RFOpenClientChannel(Device, 1);
DLCI = result[1];
if(result[0] == "Success")
{
   status = RFCreditFlowEnabled("CONNECTED_DEVICE", DLCI);
   Trace("RFCreditFlowEnabled returned: ", status, "\n\n");
}
```

RFRequestPortSettings()

RFRequestPortSettings(Address, DLCI, BaudRate, DataFormat, FlowControl, Xon, Xoff)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		Can use "CONNECTED_DEVICE" to request port settings on a master RFCOMM connection. Note that this will work only if exactly one device is connected via RFCOMM.
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()
BaudRate	String containing the baud rate		Can be "2400", "4800", "7200", "9600", "19200", "38400", "57600", "115200", "230400"
DataFormat	List of strings containing data bits, stop bits, and parity settings		Can be "RF_DATA_BITS_5", "RF_DATA_BITS_6", "RF_DATA_BITS_7", "RF_DATA_BITS_8", "RF_STOP_BITS_1", "RF_STOP_BITS_1_5", "RF_PARITY_NONE", "RF_PARITY_ON", "RF_PARITY_TYPE_ODD", "RF_PARITY_TYPE_EVEN", "RF_PARITY_TYPE_MARK", "RF_PARITY_TYPE_SPACE", "RF_DATA_BITS_MASK", "RF_STOP_BITS_MASK", "RF_PARITY_MASK", "RF_PARITY_TYPE_MASK"

Parameter	Meaning	Default Value	Comments
FlowControl	List of strings indicating port flow control options		Can be "RF_FLOW_CTRL_NONE", "RF_XON_ON_INPUT", "RF_XON_ON_OUTPUT", "RF_RTR_ON_INPUT", "RF_RTR_ON_OUTPUT", "RF_RTC_ON_INPUT", "RF_RTC_ON_OUTPUT", "RF_FLOW_RTS_CTS", "RF_FLOW_DTR_DSR", "RF_FLOW_XON_XOFF"
Xon	Number indicating the XON character		
Xoff	Number indicating the XOFF character		

Return value

- "Success"
- "Failure"
- "Not connected"
- "Timed Out"

Comments

Submits a request to change the port settings on a particular RFCOMM connection.

Example

```
result = RFOpenClientChannel(Device, 1);
DLCI = result[1];
if(result[0] == "Success")
{
   status = RFRequestPortSettings("CONNECTED_DEVICE", DLCI,
"57600", ["RF_DATA_BITS_8"], ["RF_FLOW_CTRL_NONE"], 11,
13);
   Trace("RFRequestPortSettings returned: ", status,
"\n\n");
}
```

RFRequestPortStatus()

`RFRequestPortStatus(Address, DLCI)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		Can use "CONNECTED_DEVICE" to request the port status on a master RFCOMM connection. Note that this will work only if exactly one device is connected via RFCOMM.
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()

Return value

Returns a list with two values: *status* and *portSettings*.

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Not Connected"
- "Timed Out"

portSettings (element 1) is a list containing the following five values:

- BaudRate (element 0) is a string containing the baud rate
- DataFormat (element 1) is a string containing data bits, stop bits, and parity settings
- FlowControl (element 2) is a string indicating port flow control options
- Xon (element 3) is a string containing the XON character
- Xoff (element 4) is a string containing the XOFF character

Comments

Requests the port settings on a particular RFCOMM connection.

Example

```
result = RFOpenClientChannel(Device, 1);
DLCI = result[1];
if(result[0] == "Success")
{
   res = RFRequestPortStatus(Device, DLCI);
   Trace("RFRequestPortStatus returned: ", res[0], "\n\n");
   if (res[0] == "Success")
   {
      settingsList = res[1];
      Trace("BaudRate:  ", settingsList[0], "\n");
      Trace("DataFormat:  ", settingsList[1], "\n");
      Trace("Xon:  ", settingsList[3], "\n");
```

```
            Trace("Xoff:  ", settingsList[4], "\n");
         }
      }
```

RFSetLineStatus()

`RFSetLineStatus(Address, DLCI, LineStatus)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		Can use "CONNECTED_DEVICE" to set line status on a master RFCOMM connection. Note that this will work only if exactly one device is connected via RFCOMM.
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()
LineStatus	List of strings representing the Line Status		Can be "RF_LINE_ERROR", "RF_OVERRUN", "RF_PARITY", "RF_FRAMING"

Return value

- "Success"
- "Failure"
- "Not Connected"
- "Timed Out"

Comments

Sets the line status on a particular RFCOMM connection.

Example

```
result = RFOpenClientChannel(Device, 1);
DLCI = result[1];
if(result[0] == "Success")
{
   status = RFSetLineStatus("CONNECTED_DEVICE", DLCI,
["RF_LINE_ERROR", "RF_FRAMING"]);
   Trace("RFSetLineStatus returned: ", status, "\n\n");
}
```

RFSetModemStatus()

RFSetModemStatus(Address, DLCI, ModemSignals, BreakLength)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		Can use "CONNECTED_DEVICE" to set modem status on a master RFCOMM connection. Note that this will work only if exactly one device is connected via RFCOMM.
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()
ModemSignals	List of strings specifying signal types		Can be "RF_FLOW", "RF_RTC", "RF_RTR", "RF_IC", "RF_DV", "RF_DSR", "RF_CTS", "RF_RI", "RF_CD", "RF_DTR", "RF_RTS"
BreakLength	Indicates the length of the break signal in 200 ms units		Must be between 0 and 15 (inclusive). If 0, no break signal was sent.

Return value

- "Success"
- "Failure"
- "Not Connected"
- "Timed Out"

Comments

Sets the modem status on a particular RFCOMM connection.

Example

```
result = RFOpenClientChannel(Device, 1);
DLCI = result[1];
if(result[0] == "Success")
{
   status = RFSetModemStatus("CONNECTED_DEVICE", DLCI,
["RF_FLOW"], 3);
   Trace("RFSetModemStatus returned: ", status, "\n\n");
}
```

RFSendTest()

`RFSendTest(Address, DLCI)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		Can use "CONNECTED_DEVICE" to send a test frame on a master RFCOMM connection. Note that this will work only if exactly one device is connected via RFCOMM.
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()

Return value

- "Success"
- "Failure"
- "Not Connected"
- "Failure"

Comments

Sends a test frame on a particular RFCOMM connection.

Example

```
result = RFOpenClientChannel(Device, 1);
DLCI = result[1];
if(result[0] == "Success")
{
   status = RFSendTest("CONNECTED_DEVICE", DLCI);
   Trace("RFSendTest returned: ", status, "\n\n");
}
```

RFAdvanceCredit()

`RFAdvanceCredit(Address, DLCI, credit)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device		Can use "CONNECTED_DEVICE" to advance a credit to a master RFCOMM connection. Note that this will work only if exactly one device is connected via RFCOMM.
DLCI	Data link connection identifier		The DLCI is returned by RFOpenClientChannel()

Parameter	Meaning	Default Value	Comments
credit	Number of credits to advance		

Return value

- "Success"
- "Failure"
- "Not Connected"

Comments

Advances a specified number of credits to a particular RFCOMM connection.

Example

```
result = RFOpenClientChannel(Device, 1);
DLCI = result[1];
if(result[0] == "Success")
{
   status = RFAdvanceCredit(Device, DLCI, 2);
   Trace("RFAdvanceCredit returned: ", status, "\n\n");
}
```

C.7 TCS Commands

TCSRegisterProfile()

`TCSRegisterProfile()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

Register Intercom profile with the application.

Example

```
result = TCSRegisterProfile();
Trace("TCSRegisterProfile returned: ", result, "\n");
```

TCSOpenChannel()

`TCSOpenChannel(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to connect with		

Return value

- "Success"
- "Failure"
- "Not Found"
- "Timed Out"

Comments

This command opens an L2CAP channel with TCS PSM and initializes a TCS state machine into NULL state

Example

```
result = TCSOpenChannel('010203040506');
Trace("TCSOpenChannel result : ", result, "\n");
if( result != "Success")
 return result;
```

TCSStartCall()

`TCSStartCall()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

This command must be called right after `TCSOpenChannel`. It automatically sends a sequence of TCS messages according to the Intercom profile specification of the TCS state machine. After successful execution of this command, TCS state machine is in ACTIVE state and SCO connection is opened.

Example

```
result = TCSStartCall();
Trace("TCSStartCall result : ", result, "\n");
if( result != "Success")
   return result;
```

TCSDisconnectCall()

`TCSDisconnectCall()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

This command is called to close an existing TCS connection according to the Intercom profile specification of the TCS state machine, close the L2CAP connection, and close the SCO connection.

Example

```
result = TCSDisconnectCall();
   Trace("TCSDisconnectCall result : ", result, "\n");
   if( result != "Success")
      return result;
```

TCSSendInfoMessage()

`TCSSendInfoMessage(Phone_Number)`

Parameter	Meaning	Default Value	Comments
Phone_Number	Up to 10-digit Phone Number		

Return value

- "Success"
- "Failure"
- "Invalid Parameter"

Comments

This command can be called after a TCS channel is opened. It sends an INFORMATION TCS message with a called party number.

Example

```
result = TCSSendInfoMessage("4088447081");
   Trace("TCSSendInfoMessage result : ", result, "\n");
   if( result != "Success")
      return result;

#########################################################
#  Tested TCS Call initiation, information sending,     #
#  and Call clearing                                    #
#########################################################

Main()
{
   #Device = '838010AC0008';

   Device = DoInquiry();
   Trace(Device, "\n");

   result = Connect(Device[0]);
   Trace("Connection result : ", result, "\n");
   if( result != "Success")
      return result;

   Sleep(1000);

   result = TCSRegisterProfile();
   Trace("TCSRegisterProfile result : ", result, "\n");
   if( result != "Success")
      return result;

   Sleep(1000);

   result = TCSOpenChannel(Device[0]);
   Trace("TCSOpenChannel result : ", result, "\n");
   if( result != "Success")
      return result;

   Sleep(1000);

   result = TCSStartCall();
   Trace("TCSStartCall result : ", result, "\n");
   if( result != "Success")
      return result;

   Sleep(1000);
```

```
        result = TCSSendInfoMessage("4088447081");
        Trace("TCSSendInfoMessage result : ", result, "\n");
        if( result != "Success")
           return result;

        Sleep(1000);

        result = TCSDisconnectCall();
        Trace("TCSDisconnectCall result : ", result, "\n");
        if( result != "Success")
           return result;

        Sleep(1000);

        Trace("HCI Disconnect result: ",
        Disconnect(Device[0]), "\n");
     }
```

C.8 L2CAP Commands

L2CAPConfigurationRequest()

`L2CAPConfigurationSetup(FlushTimeout, ServiceType, TokenRate, TokenBucketSize, PeakBandwidth, Latency, DelayVariation)`

Parameter	Meaning	Default Value	Comments
FlushTimeout	Amount of time that the sender will attempt transmission before flushing the packet	0xFFFF	Time is in milliseconds.
ServiceType	The required level of service	0x01	Possible values: 0x00 (no traffic), 0x01 (best effort),0x02 (guaranteed), Other (reserved)
TokenRate	The rate at which traffic credits are granted	0x00000000	0x00000000: no token rate is specified. 0xFFFFFFFF: a wild card value that matches the maximum token rate. Rate is in bytes per second.
TokenBucket Size	The size of the token bucket	0x00000000	0x00000000: no token bucket is needed. 0xFFFFFFFF: a wild card value that matches the maximum token bucket. Size is in bytes.

Parameter	Meaning	Default Value	Comments
PeakBandwidth	A value that limits the speed at which packets may be sent consecutively	0x00000000	The default value indicates that the maximum bandwidth is unknown. The speed is in bytes per second.
Latency	The maximum delay that is acceptable between transmission of a bit and its initial transmission over the air	0xFFFFFFFF	The default value represents a Do Not Care. The delay is in milliseconds.
DelayVariation	This value represents the difference between the maximum and minimum delay possible that a packet will experience	0xFFFFFFFF	The default value represents a Do Not Care. The difference is in microseconds.

Return value

- "Success"
- "Failure"

Comments

This command is used to request specified configuration for L2CAP channel. It should be executed before L2CAPConnectRequest().

For a detailed description of parameters, see the L2CAP section of the Bluetooth Specification.

Example

```
L2CAPConfigurationSetup(0xFFFF, 1, 0, 0, 0, 0xFFFFFFFF,
0xFFFFFFFF);
```

L2CAPConfigurationResponse()

`L2CAPConfigurationResponse(Reason)`

Parameter	Meaning	Default Value	Comments
Reason	Configuration response	"Accept"	Possible values: "Accept" "Reject-unknown options" "Reject-unacceptable params" "Reject"
TokenRate			
TokenBucketSize			
PeakBandWidth			
Latency			
DelayVariation			

Return value

- "Success"
- "Failure"

Comments

This command is used to automatically send the response to an incoming configuration request. It should be executed before an incoming configuration request.

Example

```
L2CAPConfigurationResponse("Reject-unknown options");
```

L2CAPConnectRequest()

`L2CAPConnectRequest(Address, PSM, ReceiveMTU)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of the remote device		
PSM			
ReceiveMTU		0x01C2	

Return value

Returns a list with three values: *result*, *ACL Handle*, and a *list of all L2CAP CIDs.*

Result (element 0) is one of the following:

- "Success"
- "Failure"
- "Not found"
- "Not connected"

ACL Handle (element 1) is a unique identifier for an ACL connection.

List of all L2CAP CIDs (element 2) is a list of all channel identifiers for an L2CAP connection with a particular device.

Comments

This command is used to establish an L2CAP channel to the specified remote device.

Example

```
result = L2CAPConnectRequest('0080370DBD02', 0x1001,
0x1C2);
Trace("L2CAPConnectRequest returned: ", result[0], "\n");
if (result[0] == "Success")
{
   Handle = result[1];
   CID = result[2];
}
```

L2CAPConnectResponse()

`L2CAPConnectResponse(Response)`

Parameter	Meaning	Default Value	Comments
Response		"Accept"	Possible values: "Accept" "Reject_Pending" "Reject_PSM_Not_Supported" "Reject_Security_Block" "Reject_No_Resources"

Return value

- "Success"
- "Failure"

Comments

This command is used to send an automatic response to an incoming L2CAP connection request. Execute this command before an incoming connection request.

Example

```
L2CAPConnectResponse("Reject_No_Resources");
```

L2CAPDeregisterAllPsm()

`L2CAPDeregisterAllPsm()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

This command is used to deregister all registered PSMs identifiers with L2CAP

Example

```
result = L2CAPDeregisterAllPsm();
Trace("DeregisterAllPsm : ", result, "\n");
```

L2CAPDisconnectRequest()

`L2CAPDisconnectRequest(CID)`

Parameter	Meaning	Default Value	Comments
CID	L2CAP channel identifier		

Return value

- "Success"
- "Failure"
- "Not connected"

Comments

This command is used to disconnect specified L2CAP channel

Example

```
L2CAPDisconnectRequest(0x0040);
```

L2CAPEchoRequest()

`L2CAPEchoRequest(Address, Data)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of the remote device		
Data			

Return value

Returns a list with two values: *status* and *data*.

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Not found"
- "Not connected"

Data (element 1) is the data returned by the remote device.

Comments

This command sends an Echo Request to the L2CAP protocol on the specified remote device. The data length should not exceed the default L2CAP signaling MTU (44 bytes).

Example

```
result = L2CAPEchoRequest('838010AC0008', "Test");
Trace("L2CAPEchoRequest result : ", result[0], "\n");

if(result[0] == "Success")
{
   Trace("Data : ", result[1], "\n");
}
```

L2CAPReceiveData()

`L2CAPReceiveData(Timeout)`

Parameter	Meaning	Default Value	Comments
Timeout	Time in ms to wait for an incoming L2CAP data	0 (Infinite wait)	Use 0 as the timeout value to wait infinitely

Return value

Returns a list with the following values: [status, number_of_bytes, data_array]

Status (element 0) is one of the following:

- "Success"
- "Timed out"
- "Not connected"

Comments

This command sets the Timeout that Merlin's Wand will wait in milliseconds for incoming L2CAP data. This function will block the specified amount of time (or infinitely if 0 is specified) unless data is received.

Example

result = L2CAPReceiveData();

Trace("L2CAPReceiveData:", result,"\n");

L2CAPInfoRequest()

`L2CAPInfoRequest(Address)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of the remote device		

Return values

Returns a list with three values: *status, number of bytes,* and *data*.

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Not found"

- "Not connected"

Number of bytes (element 1) is the number of bytes of data that follow.

Data (element 2) is the raw data.

Comments

Sends an Info Request to the L2CAP protocol on the specified remote device. Info requests are used to exchange implementation-specific information regarding L2CAP's capabilities.

Example

```
result = L2CAPInfoRequest('838010AC0008');
Trace("L2CAPInfoRequest result : ", result[0], "\n");
if(result[0] == "Success")
{
   Trace("Data length : ", result[1], "\n");
   Trace("Data        : ", result[2], "\n");
}
```

L2CAPRegisterPsm()

`L2CAPRegisterPsm(PSM, ReceiveMTU)`

Parameter	Meaning	Default Value	Comments
PSM			
ReceiveMTU		0x1C2	Incoming MTU size for L2CAP connection with that PSM

Return value

- "Success"
- "Failure"
- "In use"

Comments

This command is used to register a PSM identifier with L2CAP.

Example

```
Trace("Register PSM\n");
result = L2CAPRegisterPsm(0x1001, 0x1C2);
Trace(" Result : ", result, "\n");
```

L2CAPSendData()

`L2CAPSendData(ChannelID, Data)`

Parameter	Meaning	Default Value	Comments
ChannelID	L2CAP ChannelID to send data to		
Data	Data to send		Data can be a string, 32-bit integer value or a list containing either or both types

Return value

- "Success"
- "Timed out"
- "Not supported" (invalid data type)
- "Not connected"

Comments

An L2CAP connection must already be established with the device.

Example

```
result = L2CAPSendData(0x40, "test data");
Trace("Result : ", result, "\n");
```

L2CAPSendDataFromPipe()

`L2CAPSendDataFromPipe(ChannelID, PipeName)`

Parameter	Meaning	Default Value	Comments
ChannelID	L2CAP ChannelID to send data to		
PipeName	Name of the transmit data pipe to get data to send		This pipe must exist.

Return value

- "Success"
- "Timed out"
- "Not supported" (invalid data type)
- "Not connected"
- "Pipe not found"

Comments

An L2CAP connection must already be established with the device. The pipe specified must already be set up in the Data Transfer Manager. The pipe should not be open when L2CAPSendDataFromPipe is called.

Example

```
L2CAPSendDataFromPipe(0x0040, "Pipe1");
```

L2CAPWaitForConnection()

`L2CAPWaitForConnection(Timeout)`

Parameter	Meaning	Default Value	Comments
Timeout	Time in ms to wait for an incoming L2CAP connection	0 (Infinite wait)	Use 0 as the timeout value to wait infinitely.

Return value

- "Success"
- "Timed out"

Comments

Waits Timeout milliseconds for a device to establish an L2CAP connection with Merlin's Wand. This function will block the specified amount of time (or infinitely if 0 is specified) unless a connection is established.

Example

```
Trace("L2CAPWaitForConnection\n");
result = L2CAPWaitForConnection();
if( result == "Success")
{
   #Do something else
}
```

C.9 SDP Commands

`SDPAddProfileServiceRecord()`

`SDPAddProfileServiceRecord(ServerChannel, Profile)`

Parameter	Meaning	Default Value	Comments
Server Channel	RFCOMM server channel to accept incoming connections to this profile		Use the result from RFRegisterServerChannel() here
Profile	Name of SDP profile		Profile can be one of: "Headset", "HeadsetAudioGateway", "SerialPort", "DialUp", "FileTransfer", "Fax", "LAN", "ObjectPush", "Intercom", "Cordless", "Sync", "SyncCommand"

Return value

- "Success"
- "Failure"

Comments

Adds a profile to Merlin's Wand SDP database

Example

```
SDPAddProfileServiceRecord(rfChannel, "ObjectPush");
```

`SDPQueryProfile()`

`SDPQueryProfile(Address, Profile)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to query		
Profile	Name of SDP profile		Profile can be one of: "Headset", "HeadsetAudioGateway", "SerialPort", "DialUp", "FileTransfer", "ObjectPush", "Intercom", "Cordless", "Fax", "LAN", "GN", "PANU", "Printing", "HCRP_Server", "HCRP_Client", "Sync" or "SyncCommand"

Return value

- RFCOMM channel of the requested profile (profile is supported)
- "Failure"

Comments

Queries the specified device to see if a profile is supported.

An ACL connection must already be established with the device.

Example

```
if((RFCommId = SDPQueryProfile(Devices[0], "SerialPort"))
!= "Failure")
{
   RFOpenClientChannel(Devices[0], RFCommId);
}
```

SDPResetDatabase()

SDPResetDatabase()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

Clears all records out of the Merlin's Wand SDP profile database.

Example

```
SDPResetDatabase();
```

SDPAddServiceRecord()

SDPAddServiceRecord(FileName, RecordName, ServerChannel)

Parameter	Meaning	Default Value	Comments
FileName	String containing the full path of the file that contains the record		

Parameter	Meaning	Default Value	Comments
RecordName	String containing the name of the service record to be added		
ServerChannel	RFCOMM server channel	0	If you don't want to change the RFCOMM Server ID, set this value to 0 (or leave it blank)

Return value

- "Success"
- "Failure"
- "Failure: Could not load file"
- "Failure: Record not found"
- "Failure: Could not set RFCOMM server channel X"

Comments

If a server channel is specified, tries to set the RFCOMM server channel. If it succeeds, then it parses the file specified by FileName and tries to add the record specified by RecordName.

Example

```
status = SDPAddServiceRecord("C:\Records.sdp", "FTP Test
Record", 1);
Trace("SDPAddServiceRecord returned: ", status, "\n\n");
```

C.10 Merlin Commands

MerlinResetAllEncryptionOptions()

`MerlinResetAllEncryptionOptions()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

This command is used to remove all previously associated link keys and PIN numbers.

Example

```
MerlinResetAllEncryptionOptions();
```

MerlinSetDisplayOptions()

`MerlinSetDisplayOptions(DispOptionsFile)`

Parameter	Meaning	Default Value	Comments
DispOptionsFile	Name and full path to the display options file		

Return value

- "Success"

Comments

This command is used to set the display options file.

Example

```
MerlinSetDisplayOptions("C:\\Program
Files\\CATC\\Merlin\\default.opt");
```

MerlinSetEncryptionLinkKey()

`MerlinSetEncryptionLinkKey(Address, LinkKey)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth Address		
LinkKey	Corresponding PIN for the address		LinkKey length must be no longer than 16 bytes

Return value

- "Success"
- "Failure"
- "Invalid parameter"

Comments

This command is used to associate the LinkKey with the device address used to decrypt and display encrypted traffic.

Example

```
MerlinSetEncryptionLinkKey('008037BB2100',
"003344121222ACBBEE6172000FF0");
```

MerlinSetEncryptionPIN()

`MerlinSetEncryptionPIN(Address, PIN)`

Parameter	Meaning	Default Value	Comments
Address	Bluetooth Address		
PIN	Corresponding PIN for the address		

Return value

- "Success"
- "Failure"

Comments

This command is used to associate the PIN number with the device address used to decrypt and display encrypted traffic.

Example

```
MerlinSetEncryptionPIN('008037BB2100', "1234");
```

MerlinSetRecordingOptions()

`MerlinSetRecordingOptions(RecOptionsFile)`

Parameter	Meaning	Default Value	Comments
RecOptionsFile	Name and full path to the recording options file		

Return value

- "Success"

Comments

This command is used to set the recording options file to be used for the next recording.

Example

```
MerlinSetRecordingOptions("C:\\CATC\\Merlin
\\default.rec");
```

MerlinStart()

MerlinStart(RemoteMachine)

Parameter	Meaning	Default Value	Comments
Remote Machine	Specify a remote machine to start Merlin		RemoteMachine can be a name or an IP address

Return value

- "Success"
- "Failure"
- "Already connected"

Comments

This command is used to start Merlin on a specified remote machine or on a local machine by default.

Example

```
result = MerlinStart("192.168.2.1");
Trace("Result : ", result, "\n");
```

MerlinStartRecording()

MerlinStartRecording()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

This command is used to start a Merlin recording.

Example

```
MerlinStartRecording();
```

MerlinStop()

`MerlinStop()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"

Comments

This command is used to close the Merlin application.

Example

```
MerlinStop();
```

MerlinStopRecording()

`MerlinStopRecording()`

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"

Comments

This command is used to stop a Merlin recording.

Example

```
MerlinStopRecording();
```

Appendix D: CATC Scripting Language

CATC Scripting Language (CSL) was developed to allow users to automate test processes and provide textual output to suit specific needs. CSL is used in Merlin's Wand to write traffic-generating scripts, making it possible to automate some Bluetooth command sequences. Scripts are written, saved, and run using the Script Manager utility. Scripts' output can be viewed in the Script Log.

CSL is based on C language syntax, so anyone with a C programming background will have no trouble learning CSL. The simple, yet powerful, structure of CSL also enables less experienced users to easily acquire the basic knowledge needed to start writing custom scripts.

Features of CATC Scripting Language

- Powerful -- provides a high-level API to the Bluetooth stack while simultaneously allowing implementation of complex algorithms.
- Easy to learn and use -- has a simple but effective syntax.
- Self-contained -- needs no external tools to run scripts.
- Wide range of value types -- provides efficient and easy processing of data.
- Integrated with over 100 commands -- includes commands for HCI, L2CAP, SDP, RFCOMM, TCS, OBEX, data pipes, and the CATC Merlin Analyzer.
- General purpose -- is integrated in a number of CATC products.

D.1 Values

There are five value types that may be manipulated by a script: **integers**, **strings**, **lists**, **raw bytes**, and `null`. CSL is not a strongly typed language. Value types need not be pre-declared. Literals, variables and constants can take on any of the five value types, and the types can be reassigned dynamically.

D.2 Literals

Literals are data that remain unchanged when the program is compiled. Literals are a way of expressing hard-coded data in a script.

Integers

Integer literals represent numeric values with no fractions or decimal points. Hexadecimal, octal, decimal, and binary notation are supported:

Hexadecimal numbers must be preceded by `0x`: `0x2A, 0x54, 0xFFFFFF01`

Octal numbers must begin with `0`: `0775, 017, 0400`

Decimal numbers are written as usual: `24, 1256, 2`

Binary numbers are denoted with `0b`: `0b01101100, 0b01, 0b100000`

Strings

String literals are used to represent text. A string consists of zero or more characters and can include numbers, letters, spaces, and punctuation. An *empty string* (`""`) contains no characters and evaluates to false in an expression, whereas a non-empty string evaluates to true. Double quotes surround a string, and some standard backslash (\) escape sequences are supported.

String	Represented text
`"Quote: \"This is a string literal.\""`	`Quote: "This is a string literal."`
`"256"`	`256` **Note that this does not represent the integer 256, but only the characters that make up the number.
`"abcd!$%&*"`	`abcd!$%&*`
`"June 26, 2001"`	`June 26, 2001`
`"[ 1, 2, 3 ]"`	`[ 1, 2, 3 ]`

Escape Sequences

These are the available escape sequences in CSL:

Character	Escape Sequence	Example	Output
backslash	`\\`	`"This is a backslash: \\"`	This is a backslash: \
double quote	`\"`	`"\"Quotes!\""`	"Quotes!"
horizontal tab	`\t`	`"Before tab\tAfter tab"`	Before tab After tab
newline	`\n`	`"This is how\nto get a newline."`	This is how to get a newline.
single quote	`\'`	`"\'Single quote\'"`	'Single quote'

Lists

A list can hold zero or more pieces of data. A list that contains zero pieces of data is called an *empty list*. An empty list evaluates to false when used in an expression, whereas a non-empty list evaluates to true. List literals are expressed using the square bracket (`[]`) delimiters. List elements can be of any type, including lists.

```
[1, 2, 3, 4]
[]
["one", 2, "three", [4, [5, [6]]]]
```

Raw Bytes

Raw binary values are used primarily for efficient access to packet payloads. A literal notation is supported using single quotes:

```
'00112233445566778899AABBCCDDEEFF'
```

This represents an array of 16 bytes with values starting at `00` and ranging up to `0xFF`. The values can only be hexadecimal digits. Each digit represents a nybble (four bits), and if there are not an even number of nybbles specified, an implicit zero is added to the first byte. For example:

```
'FFF'
```

is interpreted as

```
'0FFF'
```

Null

`Null` indicates an absence of valid data. The keyword `null` represents a literal `null` value and evaluates to false when used in expressions.

```
result = null;
```

D.3 Variables

Variables are used to store information, or data, that can be modified. A variable can be thought of as a container that holds a value.

All variables have names. Variable names must contain only alphanumeric characters and the underscore (_) character, and they cannot begin with a number. Some possible variable names are

```
x
_NewValue
name_2
```

A variable is created when it is assigned a value. Variables can be of any value type, and can change type with re-assignment. Values are assigned using the assignment operator (=). The name of the variable goes on the left side of the operator, and the value goes on the right:

```
x = [ 1, 2, 3 ]
New_value = x
name2 = "Smith"
```

If a variable is referenced before it is assigned a value, it evaluates to null.

There are two types of variables: global and local.

Global Variables

Global variables are defined outside of the scope of functions. Defining global variables requires the use of the keyword `set`. Global variables are visible throughout a file (and all files that it includes).

```
set Global = 10;
```

If an assignment in a function has a global as a left-hand value, a variable will not be created, but the global variable will be changed. For example

```
set Global = 10;

Function()
{
  Global = "cat";
  Local = 20;
}
```

will create a local variable called `Local`, which will only be visible within the function `Function`. Additionally, it will change the value of `Global` to `"cat"`, which will be visible to all functions. This will also change its value type from an integer to a string.

Local Variables

Local variables are not declared. Instead, they are created as needed. Local variables are created either by being in a function's parameter list, or simply by being assigned a value in a function body.

```
Function(Parameter)
{
  Local = 20;
}
```

This function will create a local variable `Parameter` and a local variable `Local`, which has an assigned value of `20`.

D.4 Constants

A constant is similar to a variable, except that its value cannot be changed. Like variables, constant names must contain only alphanumeric characters and the underscore (_) character, and they cannot begin with a number.

Constants are declared similarly to global variables using the keyword `const`:

```
const CONSTANT = 20;
```

They can be assigned to any value type, but will generate an error if used in the left-hand side of an assignment statement later on. For instance,

```
const constant_2 = 3;

Function()
{
    constant_2 = 5;
}
```

will generate an error.

Declaring a constant with the same name as a global, or a global with the same name as a constant, will also generate an error. Like globals, constants can only be declared in the file scope.

D.5 Expressions

An expression is a statement that calculates a value. The simplest type of expression is assignment:

```
x = 2
```

The expression `x = 2` calculates 2 as the value of x.

All expressions contain operators, which are described in Section D.6, "Operators," on page 261. The operators indicate how an expression should be evaluated in order to arrive at its value. For example

```
x + 2
```

says to add 2 to x to find the value of the expression. Another example is

```
x > 2
```

which indicates that x is greater than 2. This is a Boolean expression, so it will evaluate to either true or false. Therefore, if `x = 3`, then `x > 2` will evaluate to true; if `x = 1`, it will return false.

True is denoted by a non-zero integer (any integer except 0), and false is a zero integer (0). True and false are also supported for lists (an empty list is false, while all others are true), and strings (an empty string is false, while all others are true), and null is considered false. However, all Boolean operators will result in integer values.

`select` expression

The `select` expression selects the value to which it evaluates based on Boolean expressions. This is the format for a `select` expression:

```
select {
   <expression1> : <statement1>
   <expression2> : <statement2>
   ...
};
```

The expressions are evaluated in order, and the statement that is associated with the first true expression is executed. That value is what the entire expression evaluates to.

```
x = 10
Value_of_x = select {
   x < 5 : "Less than 5";
   x >= 5 : "Greater than or equal to 5";
};
```

The above expression will evaluate to "Greater than or equal to 5" because the first true expression is `x >= 5`. Note that a semicolon is required at the end of a `select` expression because it is not a compound statement and can be used in an expression context.

There is also a keyword `default,` which in effect always evaluates to true. An example of its use is

```
Astring = select {
   A == 1 : "one";
   A == 2 : "two";
   A == 3: "three";
   A > 3 : "overflow";
   default : null;
};
```

If none of the first four expressions evaluates to true, then `default` will be evaluated, returning a value of `null` for the entire expression.

`select` expressions can also be used to conditionally execute statements, similar to C `switch` statements:

```
select {
   A == 1 : DoSomething();
   A == 2 : DoSomethingElse();
   default: DoNothing();
};
```

In this case the appropriate function is called depending on the value of A, but the evaluated result of the `select` expression is ignored.

D.6 Operators

An operator is a symbol that represents an action, such as addition or subtraction, that can be performed on data. Operators are used to manipulate data. The data being manipulated are called *operands*. Literals, function calls, constants, and variables can all serve as operands. For example, in the operation

```
x + 2
```

the variable `x` and the integer `2` are both operands, and `+` is the operator.

Operations can be performed on any combination of value types, but will result in a null value if the operation is not defined. Defined operations are listed in the Operand Types column of the table on page 264. Any binary operation on a null and a non-null value will result in the non-null value. For example, if

```
x = null;
```

then

```
3 * x
```

will return a value of 3.

A binary operation is an operation that contains an operand on each side of the operator, as in the preceding examples. An operation with only one operand is called a unary operation, and requires the use of a unary operator. An example of a unary operation is

```
!1
```

which uses the logical negation operator. It returns a value of 0.

The unary operators are `sizeof()`, `head()`, `tail()`, `~` and `!`.

Operator Precedence and Associativity

Operator rules of precedence and associativity determine in what order operands are evaluated in expressions. Expressions with operators of higher precedence are evaluated first. In the expression

```
4 + 9 * 5
```

the `*` operator has the highest precedence, so the multiplication is performed before the addition. Therefore, the expression evaluates to 49.

The associative operator `()` is used to group parts of the expression, forcing those parts to be evaluated first. In this way, the rules of precedence can be overridden. For example,

```
( 4 + 9 ) * 5
```

causes the addition to be performed before the multiplication, resulting in a value of 65.

When operators of equal precedence occur in an expression, the operands are evaluated according to the associativity of the operators. This means that if an operator's associativity is left to right, then the operations will be done starting from the left side of the expression. So, the expression

```
4 + 9 - 6 + 5
```

would evaluate to 12. However, if the associative operator is used to group a part or parts of the expression, those parts are evaluated first. Therefore,

```
( 4 + 9 ) - ( 6 + 5 )
```

has a value of 2.

In the following table, the operators are listed in order of precedence, from highest to lowest. Operators on the same line have equal precedence, and their associativity is shown in the second column.

Operator Symbol	Associativity
`[]` `()`	Left to right
`~` `!` `sizeof` `head` `tail`	Right to left
`*` `/` `%`	Left to right

Operator Symbol	Associativity
`++` `--`	Right to left
`[]` `()`	Left to right
`~` `!` `sizeof` `head` `tail`	Right to left
`*` `/` `%`	Left to right
`+` `-`	Left to right
`<<` `>>`	Left to right
`<` `>` `<=` `>=`	Left to right
`==` `!=`	Left to right
`&`	Left to right
`^`	Left to right
|	Left to right
`&&`	Left to right
||	Left to right
`=` `+=` `-=` `*=` `/=` `%=` `>>=` `<<=` `&=` `^=` |=	Right to left

Operator Symbol	Description	Operand Types	Result Types	Examples
Index Operator				

`[ ]`	Index or subscript	Raw Bytes	Integer	`Raw = '001122'`
				`Raw[1] = 0x11`

		List	Any	`List = [0, 1, 2, 3, [4, 5]]`
				`List[2] = 2`
				`List[4] = [4, 5]`
				`List[4][1] = 5`
				*Note: if an indexed Raw value is assigned to any value that is not a byte (> 255 or not an integer), the variable will be promoted to a list before the assignment is performed.

Associative Operator				

`( )`	Associative	Any	Any	`( 2 + 4 ) * 3 = 18`
				`2 + ( 4 * 3 ) = 14`

Arithmetic Operators				
`*`	Multiplication	Integer-integer	Integer	`3 * 1 = 3`
`/`	Division	Integer-integer	Integer	`3 / 1 = 3`
`%`	Modulus	Integer-integer	Integer	`3 % 1 = 0`
`+`	Addition	Integer-integer	Integer	`2 + 2 = 4`
		String-string	String	`"one " + "two" = "one two"`

		Raw byte-raw byte	Raw	`'001122' + '334455' =`
				`'001122334455'`

		List-list	List	`[1, 2] + [3, 4] = [1, 2, 3, 4]`
		Integer-list	List	`1 + [2, 3] = [1, 2, 3]`
		Integer-string	String	`"number = " + 2 = "number = 2"` *Note: integer-string concatenation uses decimal conversion.
		String-list	List	`"one" + ["two"] = ["one", "two"]`
`-`	Subtraction	Integer-integer	Integer	`3 - 1 = 2`
Increment and Decrement Operators				
`++`	Increment	Integer	Integer	*a* = 1 ++*a* = 2 *b* = 1 *b*++ = 1 *Note that the value of *b* after execution is 2.
`--`	Decrement	Integer	Integer	*a* = 2 --*a* = 1 *b* = 2 *b*-- = 2 *Note that the value of *b* after execution is 1.

Operators

Operator Symbol	Description	Operand Types	Result Types	Examples
Equality Operators				
`==`	Equal	Integer-integer	Integer	`2 == 2`
		String-string	Integer	`"three" == "three"`
		Raw byte-raw byte	Integer	`'001122' == '001122'`
		List-list	Integer	`[1, [2, 3]] == [1, [2, 3]]` *Note: equality operations on values of different types will evaluate to false.
`!=`	Not equal	Integer-integer	Integer	`2 != 3`
		String-string	Integer	`"three" != "four"`
		Raw byte-raw byte	Integer	`'001122' != '334455'`
		List-list	Integer	`[1, [2, 3]] != [1, [2, 4]]` *Note: equality operations on values of different types will evaluate to false.
Relational Operators				
`<`	Less than	Integer-integer	Integer	`1 < 2`
		String-string	Integer	`"abc" < "def"`
`>`	Greater than	Integer-integer	Integer	`2 > 1`
		String-string	Integer	`"xyz" > "abc"`
`<=`	Less than or equal	Integer-integer	Integer	`23 <= 27`
		String-string	Integer	`"cat" <= "dog"`
`>=`	Greater than or equal	Integer-integer	Integer	`2 >= 1`
		String-string	Integer	`"sun" >= "moon"` *Note: relational operations on string values are evaluated according to character order in the ASCII table.
Logical Operators				
`!`	Negation	All combinations of types	Integer	`!0 = 1   !"cat" = 0` `!9 = 0   !"" = 1`
`&&`	Logical AND	All combinations of types	Integer	`1 && 1 = 1   1 && !"" = 1` `1 && 0 = 0   1 && "cat" = 1`
`||`	Logical OR	All combinations of types	Integer	`1 || 1 = 1   0 || 0 = 0` `1 || 0 = 1   "" || !"cat" = 0`

Operators (Continued)

Operator Symbol	Description	Operand Types	Result Types	Examples
Bitwise Logical Operators				
`~`	Bitwise complement	Integer-integer	Integer	`~0b11111110 = 0b00000001`
`&`	Bitwise AND	Integer-integer	Integer	`0b11111110 & 0b01010101 = 0b01010100`
`^`	Bitwise exclusive OR	Integer-integer	Integer	`0b11111110 ^ 0b01010101 = 0b10101011`
`|`	Bitwise inclusive OR	Integer-integer	Integer	`0b11111110 | 0b01010101 = 0b11111111`
Shift Operators				
`<<`	Left shift	Integer-integer	Integer	`0b11111110 << 3 = 0b11110000`
`>>`	Right shift	Integer-integer	Integer	`0b11111110 >> 1 = 0b01111111`
Assignment Operators				

`=`	Assignment	Any	Any	`A = 1`
				`B = C = A`

`+=`	Addition assignment	Integer-integer	Integer	`x = 1`
				`x += 1 = 2`

		String-string	String	`a = "one "`
				`a += "two" = "one two"`

		Raw byte-raw byte	Raw	`z = '001122'`
				`z += '334455' = '001122334455'`

		List-list	List	`x = [1, 2]`
				`x += [3, 4] = [1, 2, 3, 4]`

		Integer-list	List	`y = 1`
				`y += [2, 3] = [1, 2, 3]`

		Integer-string	String	`a = "number = "`
				`a += 2 = "number = 2"`
				*Note: integer-string concatenation uses decimal conversion.

		String-list	List	`s = "one"`
				`s + ["two"] = ["one", "two"]`

`-=`	Subtraction assignment	Integer-integer	Integer	`y = 3`
				`y -= 1 = 2`

`*=`	Multiplication assignment	Integer-integer	Integer	`x = 3`
				`x *= 1 = 3`

`/=`	Division assignment	Integer-integer	Integer	`s = 3`
				`s /= 1 = 3`

`%=`	Modulus assignment	Integer-integer	Integer	`y = 3`
				`y %= 1 = 0`

`>>=`	Right shift assignment	Integer-integer	Integer	`b = 0b11111110`
				`b >>= 1 = 0b01111111`

`<<=`	Left shift assignment	Integer-integer	Integer	`a = 0b11111110`
				`a <<= 3 = 0b11111110000`

Operators (Continued)

Operator Symbol	Description	Operand Types	Result Types	Examples
Assignment Operators (continued)				

&=	Bitwise AND assignment	Integer-integer	Integer	`a = 0b11111110`
				`a &= 0b01010101 = 0b01010100`

^=	Bitwise exclusive OR assignment	Integer-integer	Integer	`e = 0b11111110`
				`e ^= 0b01010101 = 0b10101011`

|=	Bitwise inclusive OR assignment	Integer-integer	Integer	`i = 0b11111110`
				`i |= 0b01010101 = 0b11111111`

List Operators				

sizeof()	Number of elements	Any	Integer	`sizeof([1, 2, 3]) = 3`
				`sizeof('0011223344') = 5`
				`sizeof("string") = 6`
				`sizeof(12) = 1`
				`sizeof([1, [2, 3]]) = 2`
				*Note: the last example demonstrates that the `sizeof()` operator returns the shallow count of a complex list.

head()	Head	List	Any	`head([1, 2, 3]) = 1` *Note: the Head of a list is the first item in the list.
tail()	Tail	List	List	`tail([1, 2, 3]) = [2, 3]` *Note: the Tail of a list includes everything except the Head.

Operators (Continued)

D.7 Comments

Comments may be inserted into scripts as a way of documenting what the script does and how it does it. Comments are useful as a way to help others understand how a particular script works. Additionally, comments can be used as an aid in structuring the program.

Comments in CSL begin with a hash mark (#) and finish at the end of the line. The end of the line is indicated by pressing the Return or Enter key. Anything contained inside the comment delimiters is ignored by the compiler. Thus,

```
# x = 2;
```

is not considered part of the program. CSL supports only end-of-line comments, which means that comments can be used only at the end of a line or on their own line. It's not possible to place a comment in the middle of a line.

Writing a multi-line comment requires surrounding each line with the comment delimiters

```
# otherwise the compiler would try to interpret
# anything outside of the delimiters
# as part of the code.
```

The most common use of comments is to explain the purpose of the code immediately following the comment. For example:

```
# Add a profile if we got a server channel
if(rfChannel != "Failure")
{
   result =
SDPAddProfileServiceRecord(rfChannel,
"ObjectPush");
   Trace("SDPAddProfileServiceRecord returned ",
result, "\n");
}
```

D.8 Keywords

Keywords are reserved words that have special meanings within the language. They cannot be used as names for variables, constants or functions.

In addition to the operators, the following are keywords in CSL:

Keyword	Usage
`select`	`select` expression
`set`	define a global variable
`const`	define a constant
`return`	`return` statement
`while`	`while` statement
`for`	`for` statement
`if`	`if` statement
`else`	`if-else` statement
`default`	`select` expression
`null`	null value
`in`	input context
`out`	output context

D.9 Statements

Statements are the building blocks of a program. A program is made up of list of statements.

Seven kinds of statements are used in CSL: expression statements, if statements, if-else statements, while statements, for statements, return statements, and compound statements.

Expression Statements

An expression statement describes a value, variable, or function.

```
<expression>
```

Here are some examples of the different kinds of expression statements:

```
Value: x + 3;
Variable: x = 3;
Function: Trace ( x + 3 );
```

The variable expression statement is also called an *assignment statement*, because it assigns a value to a variable.

`if` Statements

An `if` statement follows the form

```
if <expression> <statement>
```

For example,

```
if (3 && 3) Trace("True!");
```

will cause the program to evaluate whether the expression `3 && 3` is nonzero, or True. It is, so the expression evaluates to True and the `Trace` statement will be executed. On the other hand, the expression `3 && 0` is not nonzero, so it would evaluate to False, and the statement wouldn't be executed.

`if-else` Statements

The form for an `if-else` statement is

```
if <expression> <statement1>
else <statement2>
```

The following code

```
if ( 3 - 3 || 2 - 2 ) Trace ( "Yes" );
else Trace ( "No" );
```

will cause "No" to be printed, because `3 - 3 || 2 - 2` will evaluate to False (neither `3 - 3` nor `2 - 2` is nonzero).

`while` Statements

A `while` statement is written as

```
while <expression> <statement>
```

An example of this is

```
x = 2;
while ( x < 5 )
{
  Trace ( x, ", " );
  x = x + 1;
}
```

The result of this would be

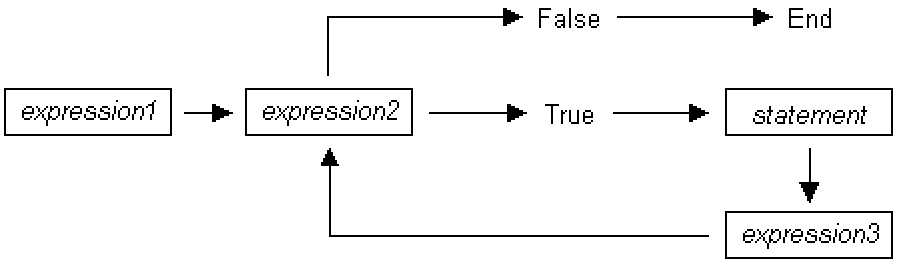
```
2, 3, 4,
```

`for` Statements

A `for` statement takes the form

```
for (<expression1>; <expression2>;
<expression3>) <statement>
```

The first expression initializes, or sets, the starting value for *x*. It is executed one time, before the loop begins. The second expression is a conditional expression. It determines whether the loop will continue -- if it evaluates true, the function keeps executing and proceeds to the statement; if it evaluates false, the loop ends. The third expression is executed after every iteration of the statement.



The example

```
for ( x = 2; x < 5; x = x + 1 ) Trace ( x, "\n" );
```

would output

```
2
3
4
```

The example above works out like this: the expression `x = 2` is executed. The value of *x* is passed to `x < 5`, resulting in `2 < 5`. This evaluates to true, so the statement `Trace (x, "\n" )` is performed, causing `2` and a new line to print. Next, the third expression is executed, and the value of *x* is increased to 3. Now, `x < 5` is executed again, and is again true, so the `Trace` statement is executed, causing 3 and a new line to print. The third expression increases the value of *x* to 4; `4 < 5` is true, so 4 and a new line are printed by the `Trace` statement. Next, the value of *x* increases to `5`. `5 < 5` is *not* true, so the loop ends.

`return` Statements

Every function returns a value, which is usually designated in a `return` statement. A `return` statement returns the value of an expression to the calling environment. It uses the following form:

```
return <expression>
```

An example of a `return` statement and its calling environment is

```
Trace ( HiThere() );
...
HiThere()
{
  return "Hi there";
}
```

The call to the primitive function `Trace` causes the function `HiThere()` to be executed. `HiThere()` returns the string "Hi there" as its value. This value is passed to the calling environment (`Trace`), resulting in this output:

```
Hi there
```

A `return` statement also causes a function to stop executing. Any statements that come after the `return` statement are ignored, because `return` transfers control of the program back to the calling environment. As a result,

```
Trace ( HiThere() );
...
HiThere()
{
  a = "Hi there";
  return a;
  b = "Goodbye";
  return b;
}
```

will output only

```
Hi there
```

because when `return a;` is encountered, execution of the function terminates, and the second return statement (`return b;`) is never processed. However,

```
Trace ( HiThere() );
```

```
...
HiThere()
{
   a = "Hi there";
   b = "Goodbye";
   if ( 3 != 3 ) return a;
   else return b;
}
```

will output

```
Goodbye
```

because the `if` statement evaluates to false. This causes the first `return` statement to be skipped. The function continues executing with the `else` statement, thereby returning the value of `b` to be used as an argument to `Trace`.

Compound Statements

A compound statement, or *statement block*, is a group of one or more statements that is treated as a single statement. A compound statement is always enclosed in curly braces ({ }). Each statement within the curly braces is followed by a semicolon; however, a semicolon is not used following the closing curly brace.

The syntax for a compound statement is

```
{
   <first_statement>;
   <second_statement>;
   ...
   <last_statement>;
}
```

An example of a compound statement is

```
{
   x = 2;
   x + 3;
}
```

It's also possible to nest compound statements, like so:

```
{
   x = 2;
   {
        y = 3;
    }
    x + 3;
}
```

Compound statements can be used anywhere that any other kind of statement can be used.

```
if (3 && 3)
{
    result = "True!";
    Trace(result);
}
```

Compound statements are required for function declarations and are commonly used in `if`, `if-else`, `while`, and `for` statements.

D.10 Preprocessing

The preprocessing command `%include` can be used to insert the contents of a file into a script. It has the effect of copying and pasting the file into the code. Using `%include` allows the user to create modular script files that can then be incorporated into a script. This way, commands can easily be located and reused.

The syntax for `%include` is this:

```
%include "includefile.inc"
```

The quotation marks around the filename are required, and by convention, the included file has a .inc extension.

The filenames given in the include directive are always treated as being relative to the current file being parsed. So, if a file is referenced via the preprocessing command in a .script file, and no path information is provided (`%include "file.inc"`), the application will try to load the file from the current directory. Files that are in a directory one level up from the current file can be referenced using `"..\file.inc"`, and likewise, files one level down can be referenced using the relative pathname (`"directory\file.inc"`). Last but not least, files can also be referred to using a full pathname, such as `"C:\global_scripts\include\file.inc"`.

D.11 Functions

A function is a named statement or a group of statements that are executed as one unit. All functions have names. Function names must contain only alphanumeric characters and the underscore (_) character, and they cannot begin with a number.

A function can have zero or more *parameters*, which are values that are passed to the function statement(s). Parameters are also known as *arguments*. Value types are not specified for the arguments or return values. Named arguments are local to the function body, and functions can be called recursively.

The syntax for a function declaration is

```
name(<parameter1>, <parameter2>, ...)
{
   <statements>
}
```

The syntax to call a function is

```
name(<parameter1>, <parameter2>, ...)
```

So, for example, a function named `add` can be declared like this:

```
add(x, y)
{
   return x + y;
}
```

and called this way:

```
add(5, 6);
```

This would result in a return value of 11.

Every function returns a value. The return value is usually specified using a `return` statement, but if no `return` statement is specified, the return value will be the value of the last statement executed.

Arguments are not checked for appropriate value types or number of arguments when a function is called. If a function is called with fewer arguments than were defined, the specified arguments are assigned, and the remaining arguments are assigned to null. If a function is called with more arguments than were defined, the extra arguments are ignored. For example, if the function `add` is called with just one argument

```
add(1);
```

the parameter x will be assigned to 1, and the parameter y will be assigned to null, resulting in a return value of 1. But if add is called with more than two arguments

```
add(1, 2, 3);
```

x will be assigned to 1, y to 2, and 3 will be ignored, resulting in a return value of 3.

All parameters are passed by value, not by reference, and can be changed in the function body without affecting the values that were passed in. For instance, the function

```
add_1(x, y)
{
  x = 2;
  y = 3;
  return x + y;
}
```

reassigns parameter values within the statements. So,

```
a = 10;
b = 20;
add_1(a, b);
```

will have a return value of 5, but the values of a and b won't be changed.

The scope of a function is the file in which it is defined (as well as included files), with the exception of primitive functions, whose scopes are global.

Calls to undefined functions are legal, but will always evaluate to null and result in a compiler warning.

D.12 Primitives

Primitive functions are called similarly to regular functions, but they are implemented outside of the language. Some primitives support multiple types for certain arguments, but in general, if an argument of the wrong type is supplied, the function will return null.

`Call()`

`Call( <function_name string>, <arg_list list> )`

Parameter	Meaning	Default Value	Comments
function_name *string*			
arg_list *list*			Used as the list of parameters in the function call.

Return value

Same as that of the function that is called.

Comments

Calls a function whose name matches the `function_name` parameter. All scope rules apply normally. Spaces in the `function_name` parameter are interpreted as the '_' (underscore) character since function names cannot contain spaces.

Example

```
Call("Format", ["the number is %d", 10]);
```

is equivalent to:

```
Format("the number is %d", 10);
```

`Format()`

`Format (<format string>, <value string or integer>)`

Parameter	Meaning	Default Value	Comments
format *string*			
value *string* or *integer*			

Return value

None.

Comments

`Format` is used to control the way that arguments will print out. The format string may contain conversion specifications that affect the way in which the arguments in the value string are returned. Format conversion characters, flag characters, and field width modifiers are used to define the conversion specifications.

Example

```
Format("0x%02X", 20);
```

would yield the string `0x14`.

`Format` can only handle one value at a time, so

```
Format("%d %d", 20, 30);
```

would not work properly. Furthermore, types that do not match what is specified in the format string will yield unpredictable results.

Format Conversion Characters

These are the format conversion characters used in CSL:

Code	Type	Output
`c`	Integer	Character
`d`	Integer	Signed decimal integer.
`i`	Integer	Signed decimal integer
`o`	Integer	Unsigned octal integer
`u`	Integer	Unsigned decimal integer
`x`	Integer	Unsigned hexadecimal integer, using "abcdef."
`X`	Integer	Unsigned hexadecimal integer, using "ABCDEF."
`s`	String	String

A conversion specification begins with a percent sign (%) and ends with a conversion character. The following optional items can be included, in order, between the % and the conversion character to further control argument formatting:

- Flag characters are used to further specify the formatting. There are five flag characters:
  - A minus sign (`-`) will cause an argument to be left-aligned in its field. Without the minus sign, the default position of the argument is right-aligned.
  - A plus sign will insert a plus sign (`+`) before a positive signed integer. This only works with the conversion characters `d` and `i`.
  - A space will insert a space before a positive signed integer. This only works with the conversion characters `d` and `i`. If both a space and a plus sign are used, the space flag will be ignored.
  - A hash mark (`#`) will prepend a `0` to an octal number when used with the conversion character `o`. If `#` is used with `x` or `X`, it will prepend `0x` or `0X` to a hexadecimal number.
  - A zero (`0`) will pad the field with zeros instead of with spaces.

- Field width specification is a positive integer that defines the field width, in spaces, of the converted argument. If the number of characters in the argument is smaller than the field width, then the field is padded with spaces. If the argument has more characters than the field width has spaces, then the field will expand to accommodate the argument.

GetNBits()

`GetNBits (<bit_source` *`list`* `or` *`raw`*`>, <bit_offset` *`integer`*`>, <bit_count` *`integer`*`>)`

Parameter	Meaning	Default Value	Comments
bit_source *list*, *raw*, or *integer*			Can be an integer value (4 bytes) or a list of integers that are interpreted as bytes.
bit_offset *integer*	Index of bit to start reading from		
bit_count	Number of bits to read		

Return value

None.

Comments

Reads `bit_count` bits from `bit_source` starting at `bit_offset`. Will return null if `bit_offset` + `bit_count` exceeds the number of bits in `bit_source`. If `bit_count` is 32 or less, the result will be returned as an integer. Otherwise, the result will be returned in a list format that is the same as the input format. `GetNBits` also sets up the bit data source and global bit offset used by `NextNBits`. Note that bits are indexed starting at bit 0.

Example

```
raw = 'F0F0';  # 1111000011110000 binary
result = GetNBits ( raw, 2, 4 );
Trace ( "result = ", result );
```

The output would be

```
   result = C     # The result is given in hexadecimal. The
result in binary is 1100
```

In the call to `GetNBits`: starting at bit 2, reads 4 bits (1100) and returns the value 0xC.

NextNBits()

`NextNBits (<bit_count integer>)`

Parameter	Meaning	Default Value	Comments
bit_count *integer*			

Return value

None.

Comments

Reads `bit_count` bits from the data source specified in the last call to `GetNBits`, starting after the last bit that the previous call to `GetNBits` or `NextNbits` returned. If called without a previous call to `GetNBits`, the result is undefined. Note that bits are indexed starting at bit 0.

Example

```
   raw = 'F0F0';  # 1111000011110000 binary
   result1 = GetNBits ( raw, 2, 4 );
   result2 = NextNBits(5);
   result3 = NextNBits(2);
   Trace ( "result1 = ", result1 " result2 = ", result2 "
result3 = ", result3 );
```

This will generate this trace output:

```
result1 = C result2 = 7 result3 = 2
```

In the call to `GetNBits`: starting at bit 2, reads 4 bits (1100), and returns the value 0xC.

In the first call to `NextNBits`: starting at bit 6, reads 5 bits (00111), and returns the value 0x7.

In the second call to `NextNBits`: starting at bit 11 (=6+5), reads 2 bits (10), and returns the value 0x2.

Resolve()

`Resolve( <symbol_name string> )`

Parameter	Meaning	Default Value	Comments
symbol_name *string*			

Return value

The value of the symbol. Returns null if the symbol is not found.

Comments

Attempts to resolve the value of a symbol. Can resolve global, constant, and local symbols. Spaces in the `symbol_name` parameter are interpreted as the '_' (underscore) character since function names cannot contain spaces.

Example

```
a = Resolve( "symbol" );
```

is equivalent to:

```
a = symbol;
```

`Trace()`

`Trace( <arg1 any>, <arg2 any>, ... )`

Parameter	Meaning	Default Value	Comments
arg *any*			The number of arguments is variable.

Return value

None.

Comments

The values given to this function are given to the debug console.

Example

```
list = ["cat", "dog", "cow"];
Trace("List = ", list, "\n");
```

would result in the output

```
List = [cat, dog, cow]
```