

**CATC BTTrainer
Bluetooth Exerciser
User's Manual
Software Version 2.20**

CATC BTTrainer 2.20 Bluetooth Exerciser User's Manual, Document Revision 1.20

Product Part Number: 730-0046-00

Document Disclaimer

The information contained in this document has been carefully checked and is believed to be reliable. However, no responsibility can be assumed for inaccuracies that may not have been detected.

CATC reserves the right to revise the information presented in this document without notice or penalty.

Trademarks and Servicemarks

CATC, *BTTracer*, *BTTrainer*, *Merlin*, and *Merlin Mobile* are trademarks of Computer Access Technology Corporation.

Bluetooth is a trademark owned by Bluetooth SIG, Inc. and is used by Computer Access Technology Corporation under license.

Microsoft, *Windows*, *Windows 98*, *Windows NT*, *Windows 2000*, *Windows ME*, and *Windows XP* are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

RadioShack is a registered trademark of RadioShack Corporation.

GN Netcom is a registered trademark of GN Netcom, Inc.

Motorola is a registered trademark of Motorola, Inc.

Belkin is a registered trademark of Belkin Components.

Coby is a registered trademark of Coby Electronics Corporation.

Plantronics is a registered trademark of Plantronics, Inc.

Intel, *Pentium*, and *Celeron* are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

AMD, *Athlon*, *Duron*, and *AMD-K6* are trademarks of Advanced Micro Devices, Inc.

All other trademarks are property of their respective companies.

Copyright

Copyright 2003, Computer Access Technology Corporation (CATC). All rights reserved.

This document may be printed and reproduced without additional permission, but all copies should contain this copyright notice.

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 complies to FCC ID KH7BT004APA-X. 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

Conformance Statements	iii
FCC Conformance Statement	iii
EU Conformance Statement	iii
Chapter 1 Overview	1
Applications	1
BTTrainer User Interface	1
Key Features	2
Audio Connections	2
Specifications	3
Package	3
Environmental Conditions	3
Host Compatibility	3
Hardware Interfaces	3
Product Warranty	3
Chapter 2 Getting Started	5
System Requirements	5
Installing the Software and Starting the Program	5
Start the program	5
Displaying the On-Screen Help	5
Application Layout	6
Menus	7
Toolbars	9
BTTrainer Toolbar	9
BTTrainer Bluetooth Analyzer Toolbar	9
Tool Tips	9
BTTrainer Keyboard Shortcuts	10
License Information	10
Chapter 3 Local Device Manager	11
General Settings	11
Device Name	11
Class of Device	12
Select as one value	12
Select components	12
Accessibility Mode	12
Connectable	12
Discoverable	13

Pairable	13
Inquiry Access Codes	13
Add an IAC	13
Modify an IAC	13
Delete an IAC	14
Security Settings	14
Security Mode	15
Bonding Mode	15
PIN Codes	15
Add a PIN Code	15
Modify a PIN Code	16
Delete a PIN Code	16
Link Keys	16
Device Information	17
Chapter 4 Profile Wizard.....	21
Starting Profile Wizard	21
Connecting to Devices	21
Connect to Device: Dial-Up Gateway	23
Connect to Device: Fax Gateway	24
Connect to Device: File Transfer	25
Connect to Device: HCRP Server	26
Connect to Device: Headset	26
Connect to Device: Headset Audio Gateway	27
Connect to Device: LAN	28
Connect to Device: Object Push	28
Connect to Device: PAN–GN	29
Connect to Device: PAN–NAP	30
Connect to Device: Serial Port	31
Emulating Devices	32
Emulate Device: Dial-Up Gateway	33
Emulate Device: Fax Gateway	34
Emulate Device: File Transfer	34
Emulate Device: HCRP Server	35
Emulate Device: Headset	35
Emulate Device: Headset Audio Gateway	36
Emulate Device: LAN	36
Emulate Device: Object Push	37
Emulate Device: PAN - GN	37
Emulate Device: PAN - NAP	38
Emulate Device: Serial Port	39
Restarting the Wizard	39
Installing the Virtual COM Port Driver	39

On Windows 2000	39
Ascertain the COM Port Number.	40
Install a Modem Driver on the Virtual COM Port.	40
Installation of Network Driver.	41
Verifying the Driver's Installation.	42
Configuration of Virtual NIC.	42
Chapter 5 Command Generator	43
Command Generator Interface	43
Using Command Generator	44
Customizing the List of HCI Commands.	45
Commands Available in Command Generator	46
HCI Commands	46
Link Control Commands	46
Link Policy Commands	47
Host Controller & Baseband Commands	48
Informational Commands	49
Status Commands	50
Testing Commands	50
CATC-Specific Commands	50
L2CAP Commands	52
SDP Commands	53
RFCOMM Commands	53
TCS Commands	54
OBEX Commands	54
BNEP Commands in Command Generator	55
TCI Commands in Command Generator	55
Chapter 6 Script Manager	57
Script Manager Interface	57
Script Manager Pop-up Menu	58
Running Scripts	58
Writing Scripts	59
Sample Scripts	60
Chapter 7 Device Search and Device List Pop-Up Menu	61
Device Search	61
Device List Pop-Up Menu	62
Create an ACL Connection	62
Establish an Audio Connection	62
Display Device Information.	63
Delete a Device	63

Disconnect All	64
Chapter 8 Contact and Warranty Information	65
Contact Information	65
Limited Hardware Warranty	65
Appendix A Command Generator Command Descriptions	69
HCI Link Control Commands	69
Accept_Connection_Request	69
Add_SCO_Connection	69
Authentication_Requested	69
Change_Connection_Link_Key	70
Change_Connection_Packet_Type	70
Create_Connection	70
Disconnect	71
Exit_Periodic_Inquiry_Mode	71
GetSCOConnections	71
Inquiry	72
Inquiry_Cancel	72
Periodic_Inquiry_Mode	72
PIN_Code_Request_Negative_Reply	73
PIN_Code_Request_Reply	73
Read_Clock_Offset	74
Read_Remote_Supported_Features	74
Read_Remote_Version_Information	74
Reject_Connection_Request	74
Remote_Name_Request	75
Set_Connection_Encryption	76
HCI Link Policy Commands	76
Get_Park_Mode	76
Exit_Sniff_Mode	76
Hold_Mode	76
Set_Park_Mode	77
QoS_Setup	77
Read_Link_Policy_Settings	78
Role_Discovery	78
Sniff_Mode	78
Switch_Role	79
Write_Link_Policy_Settings	79
HCI Host Controller & Baseband Commands	79
Change_Local_Name	79
Create_New_Unit_Key	80
Delete_Stored_Link_Key	80

Flush.....	80
Host_Buffer_Size.....	80
Read_Authentication_Enable.....	81
Read_Automatic_Flush_Timeout.....	81
Read_Class_of_Device.....	81
Read_Connection_Accept_Timeout.....	81
Read_Current_IAC_LAP.....	82
Read_Encryption_Mode.....	82
Read_Hold_Mode_Activity.....	82
Read_Inquire_Scan_Activity.....	82
Read_Link_Supervision_Timeout.....	82
Read_Local_Name.....	83
Read_Num_Broadcast_Retransmission.....	83
Read_Number_Of_Supported_IAC.....	83
Read_Page_Scan_Activity.....	84
Read_Page_Scan_Mode.....	84
Read_Page_Scan_Period_Mode.....	84
Read_Page_Timeout.....	84
Read_PIN_Type.....	85
Read_Scan_Enable.....	85
Read_SCO_Flow_Control_Enable.....	85
Read_Transmit_Power_Level.....	86
Read_Stored_Link_Key.....	86
Read_Voice_Setting.....	86
Reset.....	86
Set_Event_Filter.....	87
Set_Event_Mask.....	88
Write_Authentication_Enable.....	88
Write_Automatic_Flush_Timeout.....	89
Write_Class_of_Device.....	89
Write_Connection_Accept_Timeout.....	89
Write_Current_IAC_LAP.....	90
Write_Encryption_Mode.....	90
Write_Hold_Mode_Activity.....	91
Write_Inquiry_Scan_Activity.....	91
Write_Link_Supervision_Timeout.....	91
Write_Num_Broadcast_Retransmissions.....	92
Write_Page_Scan_Activity.....	92
Write_Page_Scan_Mode.....	92
Write_Page_Scan_Period_Mode.....	92
Write_Page_Timeout.....	92
Write_PIN_Type.....	93
Write_Scan_Enable.....	93

Write_Stored_Link_Key.....	93
Write_Voice_Setting	94
HCI Informational Commands	94
Read_BD_ADDR.....	94
Read_Buffer_Size.....	94
Read_Country_Code.....	95
Read_Local_Supported_Features.....	95
Read_Local_Version_Information.....	95
Status Commands	96
Read_Failed_Contact_Counter.....	96
Reset_Failed_Contact_Counter	96
HCI Testing Commands	96
Enable_Device_Under_Test_Mode	96
Read_Loopback_Mode.....	96
Write_Loopback_Mode	97
CATC-Specific HCI Commands	97
CATC_Change_Headset_Gain.....	97
CATC_Decrease_Power_Request	97
CATC_Disconnect	98
CATC_Get_Park_Mode.....	98
CATC_Get_Selected_Sco_Connection	99
CATC_Increase_Power_Request.....	99
CATC_MaxSlot	100
CATC_MaxSlot_Response.....	100
CATC_Modify_Beacon	101
CATC_Override_Remote_Features_Check.....	101
CATC_Page_Mode_Request	101
CATC_Page_Scan_Mode_Request	102
CATC_Qos.....	102
CATC_QoS_Response.....	103
CATC_Read_Encryption_Key_Size	103
CATC_Read_Headset_Gain.....	104
CATC_Read_Link_Key_Type.....	104
CATC_Read_PIN_Response_Enable.....	104
CATC_Read_Revision_Information	104
CATC_Sco_Parameter_Change_Response	105
CATC_Select_SCO_Connection	105
CATC_Self_Test	106
CATC_Set_Broadcast_Scan_Window.....	107
CATC_Set_Default_PIN_Code	107
CATC_Set_Park_Mode	107
CATC_Timing_Accuracy_Request	108
CATC_Write_Encryption_Key_Size.....	108

CATC_Write_Link_Key_Type	108
CATC_Write_Local_Supported_Features	109
CATC_Write_PIN_Response_Enable	109
L2CAP Command Descriptions	109
ConfigurationResponse	109
GroupRegister	110
GroupDestroy	110
GetRegisteredGroups	110
ConfigurationSetup	110
ConnectRequest	111
ConnectResponse	111
DeregisterPsm	111
DisconnectRequest	112
EchoRequest	112
InfoRequest	112
RegisterPsm	112
SendData	113
Other L2CAP Events	114
SDP Command Descriptions	114
AddProfileServiceRecord	114
AddServiceRecord	115
ProfileServiceSearch	115
RequestServiceAttribute	116
RequestServiceSearch	116
RequestServiceSearchAttribute	117
ResetDatabase	117
RFCOMM Command Descriptions	117
AcceptChannel	117
AcceptPortSettings	118
AdvanceCredit	118
CloseClientChannel	118
CreditFlowEnabled	119
DeregisterServerChannel	119
OpenClientChannel	119
RegisterServerChannel	120
RequestPortSettings	120
RequestPortStatus	121
SendData	121
SendTest	121
SetLineStatus	121
SetModemStatus	122
SendATCommand	123
Other RFCOMM Events	124

TCS Command Descriptions	124
RegisterIntercomProfile	124
Open_TCS_Channel	124
Start_TCS_Call	125
Disconnect_TCS_Call	125
Send_Info_Message	125
OBEX Command Descriptions	126
ClientAbort	126
ClientConnect	126
ClientDisconnect	126
ClientGet	126
ClientPut	127
ClientSetPath	127
ServerDeinit	127
ServerInit	128
ServerSetPath	128
BNEP Commands	129
Accept	129
Open	129
Close	129
SetUpConnectionReq	129
SentPKT	129
SendControlPKT	129
RegisterBNEP	130
DeregisterBNEP	130
SetControlTimeout	130
TCI Commands	130
CATC_EnterTestMode	130
CATC_TestControlMaster	130
CATC_CreateConnectionExt	132
CATC_AcceptConnectionExt	133
CATC_SetClock	135
CATC_SetBdAddr	135
CATC_Page	135
CATC_PageScan	136
CATC_Inquiry	136
CATC_InquiryScan	136
Appendix B BTTrainer Scripting Commands	137
Bluetooth Addresses	137
Basic Commands	137
Main	137
Clock	138

Connect	138
Disconnect	139
DoInquiry	139
GetDeviceClass	140
GetRemoteDeviceName	140
MessageBox	141
SetDeviceClass	141
Sleep	142
Pipe Commands	142
ClosePipe	142
DeletePipe	143
OpenPipe	143
ReadPipe	144
WritePipe	145
HCI Commands	146
HCIAcceptConnectionRequest	146
HCIAAddSCOConnection	146
HCIAAuthenticationRequested	147
HCICatcChangeHeadsetGain	147
HCICatcDecreasePowerRequest	149
HCICatcDisconnect	149
HCICatcGetParkMode	150
HCICATCGetSelectedSCOConnection	151
HCICATCIncreasePowerRequest	151
HCICATCModifyBeacon	152
HCICATCMaxSlot	153
HCICATCMaxSlotResponse	154
HCICATCOVERRIDERemoteFeatureCheck	155
HCICATCPageModeRequest	155
HCICATCPageScanModeRequest	156
HCICATCPageScanModeResponse	157
HCICatcQos	157
HCICATCQoSResponse	158
HCICatcReadHeadsetGain	159
HCICatcReadEncryptionKeySize	159
HCICatcReadRevisionInformation	160
HCICatcScoErrorInjection	160
HCICATCSCOParameterChangeResponse	161
HCICATCSelectSCOConnection	162
HCICatcSelfTest	162
HCICATCSetBroadcastScanWindow	163
HCICatcSetDefaultPINCode	163
HCICatcSetParkMode	164

HCICatcWriteEncryptionKeySize	166
HCICatcWriteLinkKeyType	166
HCICATCWriteLocalSupportedFeatures	167
HCICatcWritePinResponseEnable	168
HCICChangeConnectionLinkKey	168
HCICChangeConnectionPacketType	169
HCICChangeLocalName	170
HCICreateNewUnitKey	170
HCIDeleteStoredLinkKey	171
HCIEnableDeviceUnderTestMode	172
HCIExitParkMode	172
HCIExitSniffMode	173
HCIFlush	173
HCIGetSCOConnections	174
HCIIHoldMode	174
HCIMasterLinkKey	175
HCIParkMode	176
HCIPINCodeRequestNegativeReply	177
HCIPINCodeRequestReply	177
HCIQoSSetup	178
HCIReadAuthenticationEnable	179
HCIReadAutomaticFlushTimeout	180
HCIReadBDADDR	181
HCIReadBufferSize	181
HCIReadClockOffset	182
HCIReadConnectionAcceptTimeout	183
HCIReadCountryCode	184
HCIReadCurrentIACLAP	184
HCIReadEncryptionMode	185
HCIReadFailedContactCounter	186
HCIReadHoldModeActivity	186
HCIReadInquiryScanActivity	187
HCIReadLinkPolicySettings	188
HCIReadLinkSupervisionTimeout	189
HCIReadLocalName	189
HCIReadLocalSupportedFeatures	190
HCIReadLocalVersionInformation	191
HCIReadLoopbackMode	192
HCIReadNumberOfSupportedIAC	192
HCIReadNumBroadcastRetransmissions	193
HCIReadPageScanActivity	194
HCIReadPageScanMode	194
HCIReadPageScanPeriodMode	195

HCIReadPageTimeout	196
HCIReadPINType	197
HCIReadRemoteSupportedFeatures	197
HCIReadRemoteVersionInformation	198
HCIReadScanEnable	199
HCIReadSCOFlowControlEnable	200
HCIReadStoredLinkKey	200
HCIReadVoiceSetting	202
HCIRectConnectionRequest	202
HCIRemoveSCOConnection	203
HCIReset	203
HCIResetFailedContactCounter.	204
HCIRoleDiscovery.	204
HCISetConnectionEncryption	205
HCISetEventFilter	206
HCISniffMode	207
HCISwitchRole	208
HCIIWaitForMaxSlotRequest.	208
HCIIWriteAuthenticationEnable.	209
HCIIWriteAutomaticFlushTimeout	209
HCIIWriteConnectionAcceptTimeout	210
HCIIWriteCurrentIACLAP.	211
HCIIWriteEncryptionMode.	212
HCIIWriteLinkPolicySettings.	212
HCIIWriteLinkSupervisionTimeout	213
HCIIWriteLoopbackMode	214
HCIIWritePageScanActivity.	214
HCIIWritePageScanMode.	215
HCIIWritePageScanPeriodMode	216
HCIIWritePageTimeout	216
HCIIWritePINType.	217
HCIIWriteScanEnable.	217
HCIIWriteStoredLinkKey.	218
HCIIWriteVoiceSettings	218
OBEX Commands	219
OBEXClientConnect	219
OBEXClientDeinit.	220
OBEXClientDisconnect.	220
OBEXClientGet	221
OBEXClientInit	222
OBEXClientPut	222
OBEXClientSetPath.	224
OBEXServerDeinit	225

OBEXServerInit	225
OBEXServerSetPath(Path)	226
RFCOMM Commands	227
RFcloseClientChannel	227
RFOpenClientChannel	227
RFRegisterServerChannel	228
RFSendData	229
RFSendDataFromPipe	229
RFReceiveData	230
RFWaitForConnection	231
RFSendATCommand	232
RFDeregisterServerChannel	233
RFAcceptChannel	233
RFAcceptPortSettings	234
RFCreditFlowEnabled	234
RFRequestPortSettings	235
RFRequestPortStatus	236
RFSendTest()	238
RFSetLineStatus	238
RFSetCreditFlowControlEnable	239
RFSetModemStatus	240
RFSendTest	241
RFAdvanceCredit	241
TCS Commands	242
TCSRegisterProfile	242
TCSOpenChannel	243
TCSStartCall	243
TCSDisconnectCall	244
TCSSendInfoMessage	244
BNEP Commands	246
BNEPAccept()	246
BNEPClose()	247
BNEPDeregister	247
BNEPOpen()	248
BNEPRegister()	248
BNEPSendControlPkt()	249
BNEPSendPkt()	249
BNEPSendPktGeneral()	250
BNEPSetUpConnectionReq()	251
BNEPSetControlTimeout()	252
L2CAP Commands	253
L2CAPConfigurationRequest	253
L2CAPConfigurationResponse	254

L2CAPConnectRequest	255
L2CAPConnectResponse	256
L2CAPDeregisterAllPsm	256
L2CAPDisconnectRequest()	257
L2CAPEchoRequest	257
L2CAPGroupRegister	258
L2CAPGetRegisteredGroups	258
L2CAPGroupDestroy	259
L2CAPInfoRequest	260
L2CAPRegisterPsm	260
L2CAPSendData	261
L2CAPSendDataFromPipe	262
L2CAPWaitForConnection	262
SDP Commands	263
SDPAddProfileServiceRecord	263
SDPAddServiceRecord	264
SDPGenericQuery	264
SDPQueryProfile	266
SDPRequestServiceSearch	266
SDPRequestServiceAttribute	267
SDPRequestServiceSearchAttribute	269
SDPResetDatabase	269
TCI Commands	270
HCICatcAcceptConnectionExt	270
HCICatcCreateConnectionExt	273
HCICatcSetClock	276
HCICatcSetBdAddr	277
HCICatcEnterTestMode	277
HCICatcInquiry	278
HCICatcInquiryScan	279
HCICatcPage	280
HCICatcPageScan	281
HCICatcTestControlMaster	282
HCICatcEnterTestMode	289
BTTracer Commands	290
BTTracerStartRecording	290
BTTracerStopRecording	290
Appendix C CATC Scripting Language	291
Values	291
Literals	291
Integers	291
Strings	292

Lists	293
Raw Bytes	293
Null	293
Variables	293
Global Variables	294
Local Variables	294
Constants	295
Expressions	295
select expression	296
Operators	297
Comments	304
Keywords	304
Statements	305
Expression Statements	305
if Statements	306
if-else Statements	306
while Statements	306
for Statements	307
return Statements	308
Compound Statements	309
Preprocessing	310
Functions	311
Primitives	312
Call()	313
Format()	313
GetNBits()	315
NextNBits()	316
Resolve()	316
Trace()	317

CHAPTER 1: OVERVIEW

The CATC BTTrainer™ Bluetooth™ Exerciser is a member of CATC's industry-leading line of high performance, serial communication protocol analysis tools and test equipment. Preceded by CATC's BTTracer™, a Bluetooth Protocol Analyzer, BTTrainer has been designed as an intelligent Bluetooth wireless technology device that can be used as a verification and 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. BTTrainer can be used in conjunction with the BTTracer 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

BTTrainer 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, BTTrainer can be used to establish or participate in a piconet and to send or receive data within the piconet. Through BTTrainer's Profile Wizard, users can quickly and easily manage Bluetooth wireless traffic generation. Additionally, via its Command Generator, BTTrainer 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 BTTrainer User Interface

The BTTrainer 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.

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** is a point-and-click tool for creating connections and transferring data between BTTrainer 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.
- **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 maxi-

control over the traffic generation process, but also requires familiarity with the Bluetooth commands.

- **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.

1.3 Key Features

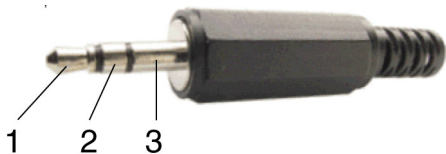
- Plug-and-play USB connection between test system and test module
- External antenna can be removed to create wired piconet
- Audio connector for connecting audio devices, such as headsets
- Can operate as either a master or slave device in a piconet
- Graphical interface allows for easy selection of 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, OBEX, BNEP, and TCI
- Scripting capability for establishment of predefined test sequences
- Support for LMP, Baseband, L2CAP, RFCOMM, SDP, BNEP test cases
- System information report provides details regarding device under test
- Works in tandem with the BTTracer analyzer to conveniently record entire session.
- Power-on self-diagnostics
- 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 www.Bluetooth.org

1.4 Audio Connections

BTTrainer 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 BTTrainer 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 BTTrainer:

- 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 Specifications

The following specifications describe a BTTrainer system.

1.5.1 Package

Connectors: Host connection (USB, type 'B')
Audio connection (2.5 millimeter audio stereo jack)

1.5.2 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

1.5.3 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.

1.5.4 Hardware Interfaces

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

1.6 Product Warranty

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

CHAPTER 2: GETTING STARTED

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

2.1 System Requirements

Please consult the readme document on the installation CD for the latest information on system requirements.

2.2 Installing the Software and Starting the Program

The BTTrainer software can be installed from the BTTracer Suite installation CD-ROM or from installation files downloaded from the CATC website.

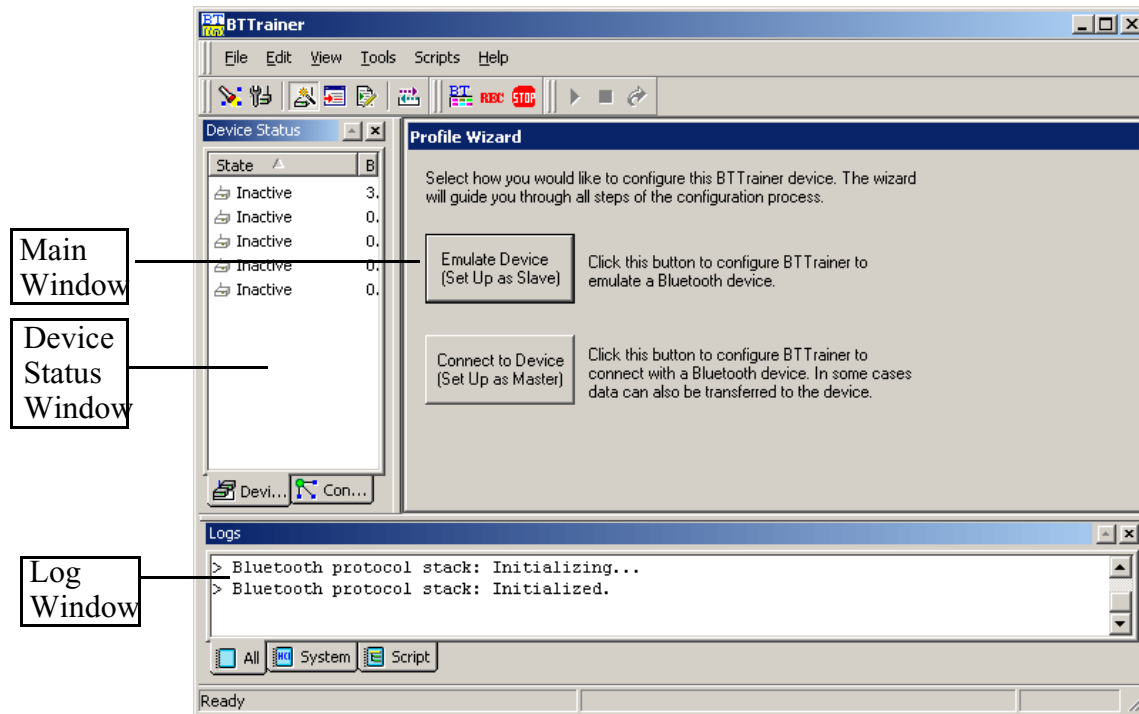
2.2.1 Start the program

The BTTrainer can be started only if the BTTracer application is active and the hardware is connected and turned on.

2.3 Displaying the On-Screen Help

Access the on-screen Help included with the BTTrainer application by selecting Help > Help... from the menu bar.

2.4 Application Layout



The BTTrainer 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 BTTrainer and other Bluetooth wireless technology devices.
 - **Command Generator** -- A tool that allows Bluetooth commands to be issued in any chosen sequence.
 - **Script Manager** -- A notepad-like tool for writing and launching scripts that cause BTTrainer to generate traffic.

Note: When switching between Profile Wizard, Command Generator and Script Manager, all connections that have been established between BTTrainer 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, BTTrainer 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 BTTrainer 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. 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 55.

- **Connections** - Displays a hierarchical list of all connections between BTTrainer and other devices. At the top of the list is the address of the connected device; below it are the various channels established between BTTrainer 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 System Log and the Script Log:
 - **System Log** - Maintains a log of the commands issued by BTTrainer and the events that ensue, such as a reply by another device.
 - **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.
 - **All Log** - Maintains a record of all commands and events.

2.5 Menus

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

Table 2-1: File Menu Commands

Command	Function
New Script	Creates a new script file
Open Script...	Opens a script file
Close Script	Closes a script file
Save Script	Saves a script file
Save Script 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 BTTrainer application

Table 2-2: Edit Menu Commands

Command	Function
Undo	Undoes last change
Cut	Cuts text
Copy	Copies text

Table 2-2: Edit Menu Commands

Command	Function
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

Table 2-3: View Menu Commands

Command	Function
BTTrainer Toolbar	Shows or hides the BTTrainer toolbar
BTTracer Analyzer Toolbar	Shows or hides the BTTracer Analyzer toolbar
Device Status	Shows or hides the Device Status window
Logs	Shows or hides the Logs window
Status Bar	Shows or hides the status bar

Table 2-4: Log Menu Commands

Command	Function
Copy Selected Log Text	Copies selected log text to the clipboard
Select All Log Text	Selects all log text in the open log window
Clear Log Window	Clears all text from the open log window
Save Log As...	Saves log file to new name and/or directory
Print Log...	Prints all text from the open log window

Table 2-5: Tools Menu Commands

Command	Function
Device Search	Opens the Device Search dialog box
Profile Wizard	Opens Profile Wizard
Command Generator	Opens Command Generator
Script Manager	Opens Script Manager
Data Transfer Manager	Opens Data Transfer Manager

Table 2-6: Help Menu Commands

Command	Function
Help...	Displays online Help
Update License...	Opens the Update License dialog, which is used to install License Keys. License Keys must be obtained from CATC.
Display License Information...	Displays maintenance expiration and features data for BTTrainer.
About BTTrainer	Displays version information about BTTrainer.

2.6 Toolbars

There are two toolbars in the BTTrainer user interface: BTTrainer toolbar and BTTracer Analyzer toolbar. The Toolbar buttons provide access to frequently-used program functions. Tool tips describe icon functionality as the mouse pointer is moved over an item.

2.6.1 BTTrainer Toolbar



Button



Action

Opens the Device Search dialog

Opens ProfileWizard

Opens Command Generator

Opens Script Manager

Opens the Data Transfer Manager dialog

Opens the Local Device Manager dialog

2.6.2 BTTrainer Bluetooth Analyzer Toolbar



Button



Action

Displays BTTracer application

Starts a BTTracer recording session

Stops a BTTracer recording session

Start Script execution

Stop Script execution

Goto - Opens a Goto dialog box for navigating to specific lines within the script

2.7 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.

2.8 BTTrainer Keyboard Shortcuts

Several frequently-used operations are bound to keyboard shortcuts.

Table 2-7: 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	Exit the application

2.9 License Information

Licensing information for BTTrainer may be viewed by selecting Help > Display License Information... from the menu bar. The License Information window will open, displaying the maintenance expiration and features data for BTTrainer.

CHAPTER 3: LOCAL DEVICE MANAGER

The Local Device Manager is a tool for viewing and configuring generic access profile settings for BTTrainer. Use the the Local Device Manager to control settings such as the Local Device Name, Class of Device, Accessibility Mode, and Security aspects.

The Local Device Manager contains three tabs: General, Security, and Device Information. The General tab is used to set the Device Name, Class of Device, Accessibility Mode, and Inquiry Access Codes. The Security tab is used to configure security settings. The Device Information tab displays details about many of the general access profile parameter settings for BTTrainer.

3.1 General Settings

The General tab contains options for generic access profile settings.

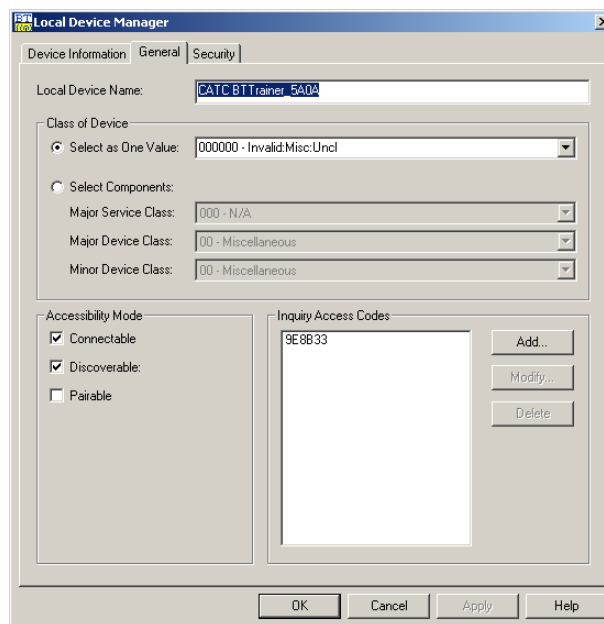


Figure 3-1: General tab of the Local Device Manager

3.1.1 Device Name

The device name is the user-friendly name by which BTTrainer identifies itself. By default, it uses the name *BTTrainer*.

To change BTTrainer's Bluetooth device name, type a new name into the text box.

3.1.2 Class of Device

The Class of Device (COD) settings define the Bluetooth Device Class and Bluetooth Service Type with which BTTrainer will identify itself in response to inquiries from other Bluetooth devices.

Select as one value

Use “Select as one value” to choose a pre-configured set of COD parameters.

To select COD settings as one value:

- Step 1 Select the radio button marked “Select as one value.”
- Step 2 Choose a set of COD parameters from the drop-down list or type in a COD parameter value.
- Step 3 Click Apply to apply the settings, or click OK to apply the settings and close the Local Device Manager.

Select components

Use “Select components” to choose the Major Service Class, Major Device Class, and Minor Device Class parameters individually.

To select COD settings as individual component values:

- Step 1 Select the radio button marked “Select components.”
- Step 2 Choose a parameter from the Major Service Class drop-down list.
- Step 3 Choose a parameter from the Major Device Class drop-down list.
- Step 4 Choose a parameter from the Minor Service Class drop-down list.
- Step 5 Click Apply to apply the settings, or click OK to apply the settings and close the Local Device Manager.

3.1.3 Accessibility Mode

Accessibility Mode is used to set the Connectable, Discoverable, and Pairable Modes for BTTrainer.

Connectable

BTTrainer can be put into connectable or non-connectable mode.

- Put BTTrainer in connectable mode by checking the Connectable checkbox and clicking Apply to apply the settings, or OK to apply the settings and close the Local Device Manager.

This causes BTTrainer to respond to paging from other Bluetooth devices.

- Put BTTrainer in non-connectable mode by unchecking the Connectable checkbox and clicking Apply to apply the settings, or OK to apply the settings and close the Local Device Manager.

This prevents BTTrainer from responding to paging from other Bluetooth devices.

Discoverable

BTTrainer can be placed in discoverable or non-discoverable mode.

- Place BTTrainer in discoverable mode by checking the Discoverable checkbox, and clicking Apply to apply the settings, or OK to apply the settings and close the Local Device Manager.
- Place BTTrainer in non-discoverable mode by unchecking the Discoverable checkbox and clicking Apply to apply the settings, or OK to apply the settings and close the Local Device Manager.

BTTrainer will not respond to inquiries from other devices while in non-discoverable mode.

Pairable

BTTrainer can put in pairable or non-pairable mode.

- Put BTTrainer in pairable mode by checking the Pairable checkbox, and clicking Apply to apply the settings, or OK to apply the settings and close the Local Device Manager.

BTTrainer will accept pairing, or bonding, initiated by a remote device while in pairable mode.

3.1.4 Inquiry Access Codes

The Inquiry Access Codes section is used to define the Inquiry Access Codes (IACs) that BTTrainer will scan for when performing an inquiry.

Add an IAC

Step 1 Click Add.

The Add IAC dialog will open.

Step 2 Enter an IAC in the text box.

Values must be entered in hex, and they must be between 9E8B00–9E8B3F.

Step 3 Click OK.

The new value will be added to the list of IACs in the Local Device Manager.

Step 4 Click Apply to apply the settings, or click OK to apply the settings and close the Local Device Manager.

Modify an IAC

Step 1 Select the IAC that you want to modify.

Note: You cannot modify the GIAC (0x9E8B33).

Step 2 Click Modify.

The Modify IAC dialog will open.

- Step 3** Edit the IAC in the text box.
Values must be entered in hex, and they must be between 9E8B00–9E8B3F.
- Step 4** Click OK.
The new value will be displayed in the list of IACs in the Local Device Manager.
- Step 5** Click Apply to apply the settings, or click OK to apply the settings and close the Local Device Manager.

Delete an IAC

- Step 1** Select the IAC that you want to delete.
Note: You cannot delete the GIAC (0x9E8B33).
- Step 2** Click Delete.
The value will be removed from the list of IACs in the Local Device Manager.
- Step 3** Click Apply to apply the settings, or click OK to apply the settings and close the Local Device Manager.

3.2 Security Settings

The Security tab contains options for security settings.

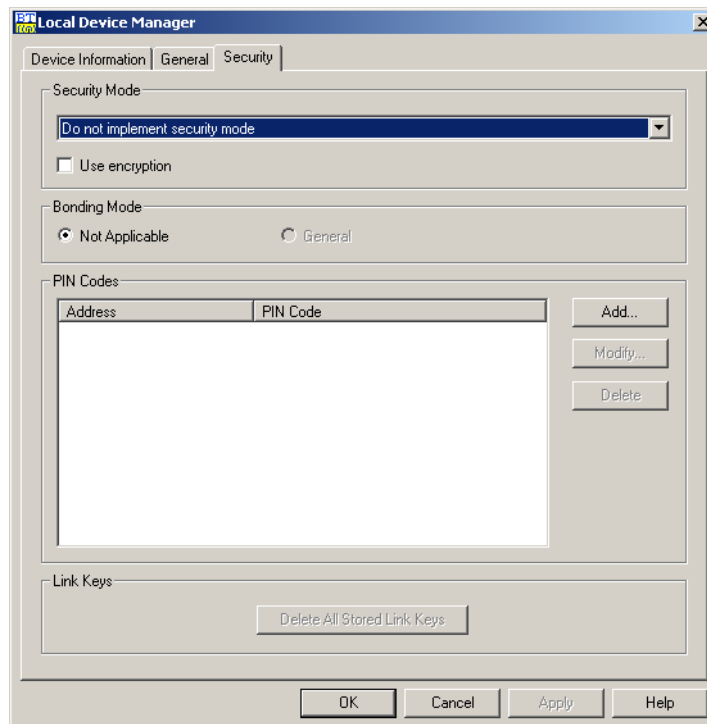


Figure 3-2: Security tab of the Local Device Manager

3.2.1 Security Mode

There are three Bluetooth security modes: security mode 1, security mode 2, and security mode 3. Another option is not to use a security mode at all.

- Security mode 1 is the non-secure mode. When in this mode, devices do not initiate security procedures.
- Security mode 2 is service-level enforced security. Devices in this mode initiate security procedures after L2CAP channel establishment has occurred.
- Security mode 3 is link-level enforced security. A device in this mode will initiate security procedures before L2CAP channel establishment has occurred.

To set the security mode that BTTrainer will use:

- Step 1** Select a security mode from the drop-down list.
- Note:** Security mode 2 is not currently supported in BTTrainer.
- Step 2** (Optional) Check or uncheck “Use encryption” to configure whether BTTrainer will implement encryption when communicating with other devices.
- Step 3** Click Apply to apply the settings, or click OK to apply the settings and close the Local Device Manager.

3.2.2 Bonding Mode

Bluetooth bonding causes two devices to create, exchange, and store a common link key. The devices store the link key and use it for authentication during future connections.

BTTrainer supports only the general bonding mode. BTTrainer sets the bonding mode based on the security mode that has been selected.

Bonding is applied only if security mode 3 has been selected in the Local Device Manager.

3.2.3 PIN Codes

You may define the PIN codes that BTTrainer should use for specific devices.

Add a PIN Code

Note: PIN codes may also be added by specifying them when using Profile Wizard to connect to a device. See Section 4.2, “Connecting to Devices” on page 21 for more information.

- Step 1** Click Add.
- The Add PIN Code dialog will open.
- Step 2** Enter the Bluetooth device address (BD_ADDR) in the first text box. It must be entered in hexadecimal, containing from 1-12 digits.
- Note:** To enter a default PIN code, enter the word “Default” instead of a BD_ADDR.

- Step 3** Enter a PIN code in the second text box. Values must contain from 1-16 characters.
- Step 4** Click OK.
The new address and PIN code will be added to the list in the Local Device Manager.
- Step 5** Click Apply to apply the settings, or click OK to apply the settings and close the Local Device Manager.

Modify a PIN Code

- Step 1** Click Modify.
The Modify PIN Code dialog will open.
- Step 2** Edit the PIN code in the text box. Values must contain from 1-16 characters.
- Step 3** Click OK.
The new value will be displayed in the list in the Local Device Manager.
- Step 4** Click Apply to apply the settings, or click OK to apply the settings and close the Local Device Manager.

Delete a PIN Code

Note: The default PIN code cannot be deleted while BTTrainer is using security mode 3.

- Step 1** Select the PIN code that you want to delete.
- Step 2** Click Delete.
The value will be removed from the list of PIN codes in the Local Device Manager.
- Step 3** Click Apply to apply the settings, or click OK to apply the settings and close the Local Device Manager.

3.2.4 Link Keys

If BTTrainer has stored any link keys, you may delete them via the Local Device Manager.

To delete stored link keys:

- Step 1** Click the Delete All Stored Link Keys button.
- Step 2** Click OK in the pop-up message box to confirm the deletion.

3.3 Device Information

The Device Information tab displays general access profile settings for BTTrainer.

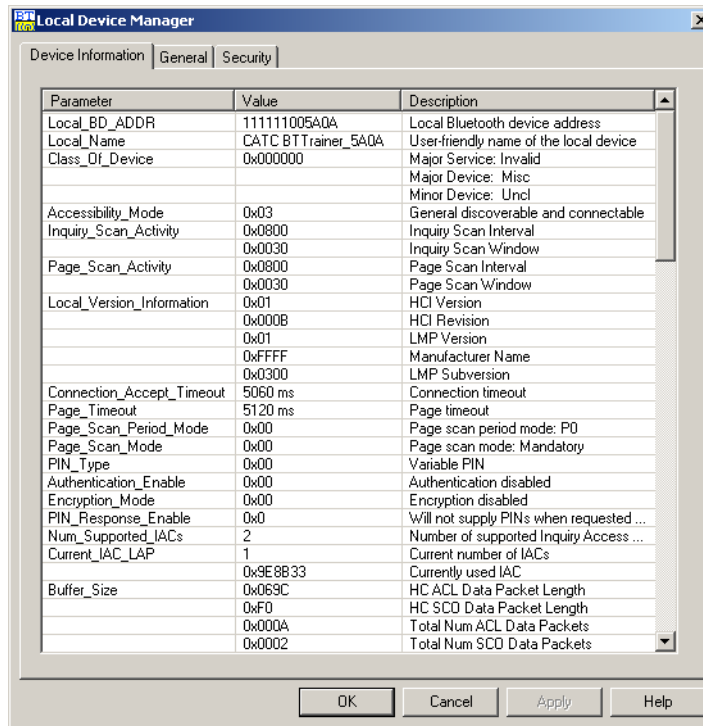


Figure 3-3: Device Information tab of the Local Device Manager

These parameter values are displayed on the Device Information tab of the Local Device Manager:

- **Local_BD_ADDR** — The Bluetooth device address for BTTrainer. Every device has a 48-bit address (BD_ADDR) that uniquely identifies it.
- **Local_Name** — The user-friendly name for BTTrainer.
- **Class_of_Device** — This parameter indicates the Bluetooth Device Class and Bluetooth Service Type with which BTTrainer will identify itself in response to inquiries from other Bluetooth devices. The Major Service Class, Major Device Class, and Minor Device Class are all included in this parameter.
- **Accessibility_Mode** — This value defines the discoverability and connectability modes for BTTrainer.
- **Local_Version_Information** — The values of the version information for BTTrainer.
 - **HCI Version:** Indicates the Bluetooth HCI Specification used by BTTrainer. Can be 0x00 (Bluetooth HCI Specification 1.0B), 0x01 (Bluetooth HCI Specification 1.1), or 0x02-0xFF (reserved for future use).
 - **HCI Revision:** The Current HCI revision in the BTTrainer hardware.

- LMP Version: The Current Link Manager Protocol version (VersNr) in the BTTrainer hardware. Can be 0 (Bluetooth LMP 1.0), 1 (Bluetooth LMP 1.1), or 2-255 (reserved for future use)
- Manufacturer Name: The name of the company (CompID) that created the LMP Subversion.
- LMP Subversion: The Current Link Manager Protocol subversion in the BTTrainer hardware.
- Connection_Accept_Timeout — The value of the Connection_Accept_Timeout parameter, which specifies the amount of time that BTTrainer will wait to accept a connection after sending a connection request.
- Page_Timeout — The value of the Page_Reply_Timeout parameter, which specifies the amount of time that BTTrainer will wait for a response to a connection request before returning a connection failure.
- Page_Scan_Period_Mode — The value of the mandatory Page_Scan_Period_Mode parameter, which specifies the value for the T_mandatory_pscan timer. Possible values:
 - 0x00: PO
 - 0x01: P1
 - 0x02: P2
 - 0x03–0xFF: Reserved
- Page_Scan_Mode — The value of the Page_Scan_Mode parameter, which is the default page scan mode for BTTrainer. Possible values:
 - 0x00: Mandatory Page Scan Mode
 - 0x01: Optional Page Scan Mode I
 - 0x02: Optional Page Scan Mode II
 - 0x03: Optional Page Scan Mode III
 - 0x04–0xFF: Reserved
- PIN_Type — The PIN type specified in the BTTrainer host controller. Possible values:
 - 0x00: Variable PIN
 - 0x01: Fixed PIN
- Authentication_Enable — The value of the Authentication_Enable parameter, which specifies whether BTTrainer will try to authenticate a remote device at connection setup. Possible values:
 - 0x00: Authentication disabled
 - 0x01: Authentication enabled for all connections
 - 0x02–0xFF: Reserved
- Encryption_Mode — The value of the Encryption_Mode parameter, which specifies whether BTTrainer requires encryption for connections with remote devices. Possible values:

- 0x00: Encryption disabled
- 0x01: Encryption only for point-to-point packets
- 0x02: Encryption for both point-to-point and broadcast packets
- 0x03–0xFF: Reserved
- PIN_Response_Enable — Specifies whether BTTrainer will supply PINs when requested during authentication. Possible values:
 - 0x00: Device will not supply PINs when requested during authentication.
 - 0x01: Device will supply PINs when requested during authentication.
- Num_Supported_IACs — The number of Inquiry Access Codes (IACs) that BTTrainer can listen for at the same time during an Inquiry Scan.
- CBuffer_Size — The maximum size of HCI SCO and ACL data packets supported by the BTTrainer Host Controller, as well the the maximum number of such packets that BTTrainer can store in its data buffer. Included parameters:
 - HC_ACL_Data_Packet_Length: The maximum length, in bytes, of HCI ACL data that BTTrainer can accept in one packet.
 - HC_SCO_Data_Packet_Length: The maximum length, in bytes, of HCI SCO data that BTTrainer can accept in one packet.
 - HC_Total_Num_ACL_Data_Packets: The maximum number of HCI ACL data packets that the BTTrainer Host Controller can store in its data buffer.
 - HC_Total_Num_SCO_Data_Packets: The maximum number of HCI SCO data packets that the BTTrainer Host Controller can store in its data buffer.
- Local_Supported_Features — A list of the supported features for BTTrainer.

CHAPTER 4: PROFILE WIZARD


Once BTTrainer 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 BTTrainer 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 BTTrainer will execute the Bluetooth commands needed to make the connection.

Profile Wizard allows you to configure BTTrainer 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

4.1 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.

4.2 Connecting to Devices

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

Note: 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 BTTrainer 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.

BTTrainer 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 BTTrainer will not try to discover the devices’ user-friendly names. This will speed up the search.

Note: If no devices are found, you can click **Search** to have BTTrainer repeat the General Inquiry.

Step 3 *Select device from list*

From the list, select the device address to which you want BTTrainer 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**.

BTTrainer 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 4.2.1, “Connect to Device: Dial-Up Gateway” on page 23
- Section 4.2.2, “Connect to Device: Fax Gateway” on page 24
- Section 4.2.3, “Connect to Device: File Transfer” on page 25
- Section 4.2.4, “Connect to Device: HCRP Server” on page 26
- Section 4.2.5, “Connect to Device: Headset” on page 26
- Section 4.2.6, “Connect to Device: Headset Audio Gateway” on page 27
- Section 4.2.7, “Connect to Device: LAN” on page 28
- Section 4.2.8, “Connect to Device: Object Push” on page 28
- Section 4.2.9, “Connect to Device: PAN–GN” on page 29
- Section 4.2.10, “Connect to Device: PAN–NAP” on page 30
- Section 4.2.11, “Connect to Device: Serial Port” on page 31

4.2.1 Connect to Device: Dial-Up Gateway

Follow these steps to configure BTTrainer to connect to a Bluetooth device that uses the Dial-Up Gateway profile.

Note: To connect to a device that uses the Dial-Up Gateway profile, you must install the virtual COM port driver that is included with the BTTrainer installation. For instructions, see “Installing the Virtual COM Port Driver” on page 39.

Step 1 Complete steps 1-6 in Section 4.2, “Connecting to Devices” on page 21.

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 BTTrainer to switch roles during the connection.

Step 3 Click **Next**.

BTTrainer 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:

Note: A modem driver must be installed on the virtual COM port in order to use an external application to send data through BTTrainer. 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.

Note: An 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 BTTrainer.

Enable or disable event logging:

You can configure BTTrainer to write incoming or outgoing data to the Event Log by checking or unchecking the “Log Incoming Data” and “Log Outgoing Data” options.

4.2.2 Connect to Device: Fax Gateway

The Fax Gateway option allows you to use a fax application to send data to BTTrainer via a virtual COM port. BTTrainer 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 BTTrainer installation. For instructions, see “Installing the Virtual COM Port Driver” on page 39.

Step 1 Complete steps 1-6 in Section 4.2, “Connecting to Devices” on page 21.

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 BTTrainer to switch roles during the connection.

Step 3 Click **Next**.

BTTrainer 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:

Note: A modem driver must be installed on the virtual COM port in order to use a fax application to send data through BTTrainer. 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.

Note: A 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 BTTrainer.

Enable or disable event logging:

You can configure BTTrainer to write incoming or outgoing data to the Event Log by checking or unchecking the “Log Incoming Data” and “Log Outgoing Data” options.

4.2.3 Connect to Device: File Transfer

These steps show how to configure BTTrainer 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 4.2, “Connecting to Devices” on page 21.

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 BTTrainer to switch roles during the connection.

Step 3 Click **Next**.

BTTrainer 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 (BTTrainer):

- **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.

4.2.4 Connect to Device: HCRP Server

This section explains how to use Profile Wizard to configure BTTrainer 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 4.2, "Connecting to Devices" on page 21.

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 BTTrainer to switch roles during the connection.

Step 3 Press **Next**.

BTTrainer 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.

4.2.5 Connect to Device: Headset

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

Note: In order to verify that BTTrainer and a Bluetooth audio device are successfully connected, a headset will need to be plugged into the audio port on BTTrainer. Be sure that the headset is plugged in *before* starting configuration and initializing the connection between BTTrainer and the headset. To learn about attaching a headset to BTTrainer, please see "Audio Connections" on page 2.

Step 1 Complete steps 1-6 in Section 4.2, "Connecting to Devices" on page 21.

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

(Optional) Select the Allow Role Switch option on the screen to enable BTTrainer to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen, and BTTrainer will attempt to establish an RFCOMM connection to the device.

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

When the target device answers, BTTrainer 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 BTTrainer and the device will terminate, and the **Connect** button will again be available. Selecting Connect will reestablish the connection.

4.2.6 Connect to Device: Headset Audio Gateway

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

Note: In order to verify that BTTrainer and a Bluetooth audio device are successfully connected, a headset will need to be plugged into the audio port on BTTrainer. Be sure that the headset is plugged in *before* starting configuration and initializing the connection between BTTrainer and the headset. To learn about attaching a headset to BTTrainer, please see “Audio Connections” on page 2.

Step 1 Complete steps 1-6 in Section 4.2, “Connecting to Devices” on page 21.

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 BTTrainer to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen, and BTTrainer will attempt to establish a connection to the device.

If the connection attempt is successful, the Connection Status screen will show that BTTrainer 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 BTTrainer 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 BTTrainer and the device will terminate, and the **Connect** button will again be available. Selecting Connect will reestablish the connection.

4.2.7 Connect to Device: LAN

This section explains how to configure BTTrainer 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 BTTrainer installation. For instructions, See “Installing the Virtual COM Port Driver” on page 39.

Step 1 Complete steps 1-6 in Section 4.2, “Connecting to Devices” on page 21.

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

(Optional) Select the Allow Role Switch option on the screen to enable BTTrainer to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen.

4.2.8 Connect to Device: Object Push

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

Step 1 Complete steps 1-6 in Section 4.2, “Connecting to Devices” on page 21.

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 BTTrainer 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, BTTrainer 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.

4.2.9 Connect to Device: PAN–GN

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

Note: To connect to a device that uses the PAN profile, you must install the virtual network driver that is included with the BTTrainer installation. For instructions, see Section 4.7, “Installation of Network Driver,” on page 41.

Step 1 Complete steps 1-6 in Section 4.2, “Connecting to Devices” on page 21.

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 BTTrainer to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen. BTTrainer will attempt to establish an ACL connection to the remote device, and then open a BNEP connection to it. BTTrainer will then send a Setup Connection Request Message. When BTTrainer 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 *0xnnnn*. For the Filter Multicast Address Set Message, the Start and End Ranges can be in the format *nnnnnnnnnnnnnnnn* or *0xnnnnnnnnnnnnnnnn*.

- **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.'

4.2.10 Connect to Device: PAN–NAP

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

Note: To connect to a device that uses the PAN profile, you must install the virtual network driver that is included with the BTTrainer installation. For instructions, see Section 4.7, "Installation of Network Driver," on page 41.

Step 1 Complete steps 1-6 in Section 4.2, "Connecting to Devices" on page 21.

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 BTTrainer to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen. BTTrainer will attempt to establish an ACL connection to the remote device, and then open a BNEP connection to it. BTTrainer will then send a Setup Connection Request Message. When BTTrainer 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 *0xnnnn*. For the Filter Multicast Address Set Message, the Start and End Ranges can be in the format *nnnnnnnnnnnnnn* or *0xnnnnnnnnnnnnnn*.

- **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.'

4.2.11 Connect to Device: Serial Port

These steps describe how to use Profile Wizard to configure BTTrainer 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 BTTrainer installation. For instructions, see "Installing the Virtual COM Port Driver" on page 39.

Step 1 Complete steps 1-6 in Section 4.2, "Connecting to Devices" on page 21.

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 BTTrainer to switch roles during the connection.

Step 3 Click **Next**.

The Wizard will advance to the Connection Status screen, and BTTrainer will attempt to establish a connection to the device.

If the connection attempt is successful, the Connection Status screen will show that BTTrainer 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 BTTrainer. 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 BTTrainer.

4.3 Emulating Devices

Choosing the “Emulate Device” option on the opening screen of Profile Wizard allows you to configure BTTrainer to emulate a Bluetooth wireless device. This option causes BTTrainer 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 4.3.1, “Emulate Device: Dial-Up Gateway” on page 33
- Section 4.3.2, “Emulate Device: Fax Gateway” on page 34
- Section 4.3.3, “Emulate Device: File Transfer” on page 34
- Section 4.3.4, “Emulate Device: HCRP Server” on page 35
- Section 4.3.5, “Emulate Device: Headset” on page 35

- Section 4.3.6, “Emulate Device: Headset Audio Gateway” on page 36
- Section 4.3.7, “Emulate Device: LAN” on page 36
- Section 4.3.8, “Emulate Device: Object Push” on page 37
- Section 4.3.9, “Emulate Device: PAN - GN” on page 37
- Section 4.3.10, “Emulate Device: PAN - NAP” on page 38
- Section 4.3.11, “Emulate Device: Serial Port” on page 39

4.3.1 Emulate Device: Dial-Up Gateway

These steps show how to use Profile Wizard to configure BTTrainer to emulate a dial-up gateway device.

Step 1 Complete Steps 1-4 in Section 4.3, “Emulating Devices” on page 32.

The Select Dial-Up Emulation screen will open.

Step 2 You may choose to configure BTTrainer 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 BTTrainer 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 BTTrainer 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 BTTrainer 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:

Note: To connect to a virtual COM port, you will need to install the virtual COM port driver that is included with the BTTrainer 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 BTTrainer 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 BTTrainer.

4.3.2 Emulate Device: Fax Gateway

The following steps show how to configure BTTrainer to emulate a fax gateway device.

Step 1 Complete Steps 1-4 in Section 4.3, “Emulating Devices” on page 32.

The select Fax Emulation screen will open.

Step 2 You may choose between configuring BTTrainer 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 BTTrainer 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 BTTrainer 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:

Note: To connect to a virtual COM port, you will need to install the virtual COM port driver that is included with the BTTrainer 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 BTTrainer will support during emulation.

(b) Press **Next**.

At this point, initialize a connection to BTTrainer 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 BTTrainer.

4.3.3 Emulate Device: File Transfer

Follow these steps to configure BTTrainer to emulate a Bluetooth device that supports the File Transfer profile.

Step 1 Complete Steps 1-4 in Section 4.3, “Emulating Devices” on page 32.

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 BTTrainer.

An OBEX connection will be established between the device and BTTrainer.

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

4.3.4 Emulate Device: HCRP Server

This section explains how to use Profile Wizard to set up BTTrainer to emulate an HCRP server.

Step 1 Complete Steps 1-4 in Section 4.3, “Emulating Devices” on page 32.

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.

4.3.5 Emulate Device: Headset

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

Step 1 Complete Steps 1-4 in Section 4.3, “Emulating Devices” on page 32.

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

Note: The 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 BTTrainer.

Once the connection is established, the Emulation Status screen will indicate that BTTrainer 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 successful, 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 BTTrainer and the device will terminate.

4.3.6 Emulate Device: Headset Audio Gateway

BTTrainer 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 BTTrainer as a Headset Audio Gateway device and connect to it with a remote Bluetooth headset.

Step 1 Complete Steps 1-4 in Section 4.3, “Emulating Devices” on page 32.

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

Note: The 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 BTTrainer.

Once the connection is established, the Emulation Status screen will indicate that BTTrainer is currently connected to the device.

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

4.3.7 Emulate Device: LAN

Step 1 Complete Steps 1-4 in Section 4.3, “Emulating Devices” on page 32.

Step 2 In the Emulate Device dialog, select LAN.

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

4.3.8 Emulate Device: Object Push

BTTrainer 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 BTTrainer.

Step 1 Complete Steps 1-4 in Section 4.3, “Emulating Devices” on page 32.

The Emulation Status screen will open, indicating that BTTrainer 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 BTTrainer is waiting for a connection after having successfully received the file.

4.3.9 Emulate Device: PAN - GN

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

Note: To connect to a device that uses the PAN profile, you must install the virtual network driver that is included with the BTTrainer installation. For instructions, see Section 4.7, “Installation of Network Driver,” on page 41.

Step 1 Complete Steps 1-4 in Section 4.3, “Emulating Devices” on page 32.

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

The Emulation Status screen will open, indicating that BTTrainer 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 *0xnnnn*. For the Filter Multicast Address Set Message, the Start and End Ranges can be in the format *nnnnnnnnnnnnnn* or *0xnnnnnnnnnnnnnn*.

- **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.'

4.3.10 Emulate Device: PAN - NAP

Note: To connect to a device that uses the PAN profile, you must install the virtual network driver that is included with the BTTrainer installation. For instructions, see Section 4.7, "Installation of Network Driver," on page 41.

Step 1 Complete Steps 1-4 in Section 4.3, "Emulating Devices" on page 32.

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

The Emulation Status screen will open, indicating that BTTrainer 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 *0xnnnn*. For the Filter Multicast Address Set Message, the Start and End Ranges can be in the format *nnnnnnnnnnnnnn* or *0xnnnnnnnnnnnnnn*.

- **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.'

4.3.11 Emulate Device: Serial Port

These steps describe how to use Profile Wizard to configure BTTrainer to emulate a device that uses the Serial Port profile.

Note: To emulate a device that uses the Serial Port profile, you will need to install the virtual COM port driver that is included with the BTTrainer installation. For instructions, see “Installing the Virtual COM Port Driver” on page 39.

Step 1 Complete Steps 1-4 in Section 4.3, “Emulating Devices” on page 32.

The Serial Port Emulation screen will open.

Step 2 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 BTTrainer. 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 BTTrainer.

4.4 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 BTTrainer is currently emulating a connected device, the connection will be broken, and Profile Wizard will return to the default Profile Wizard screen.
- If BTTrainer 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.

4.5 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 BTTrainer installation.

4.5.1 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 BTTrainer 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.

4.6 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 BTTrainer, you need to install a modem driver on the virtual COM port.

On Windows 2000:

- Step 1** Complete steps 1-3 in Section 4.2.1, “Connect to Device: Dial-Up Gateway” on page 23 or in Section 4.2.2 on page 24.
BTTrainer 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**.
Note: To 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.

4.7 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**.

4.7.1 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.”

Note: The MAC address of the NIC driver is based on the BD address of BTTrainer. The MAC addr will be rewritten every time the BNEP_Register is executed.

4.7.2 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.

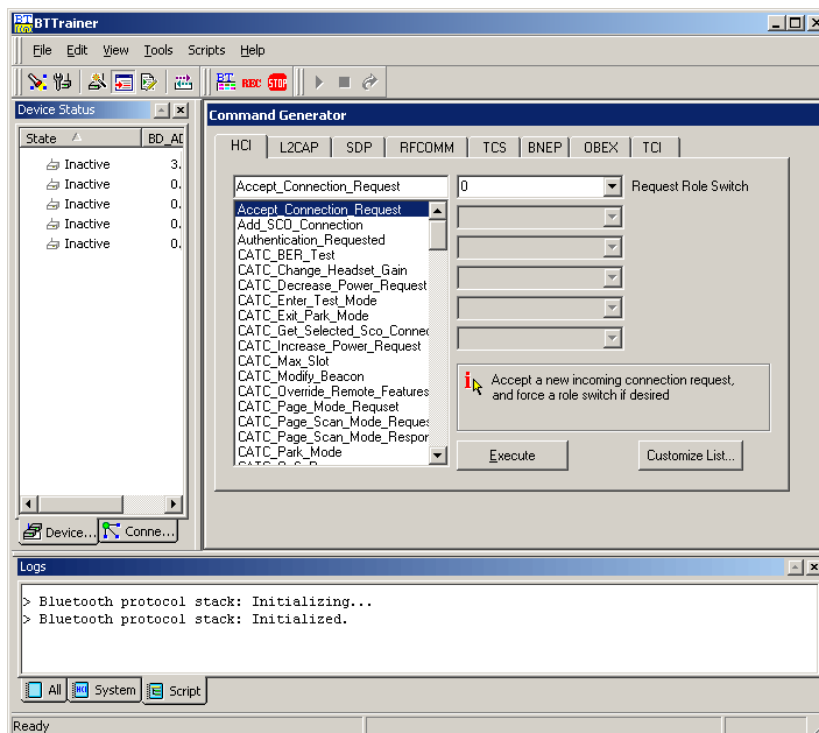
CHAPTER 5: COMMAND GENERATOR

The Command Generator is a tool in BTTrainer 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 includes Baseband, LMP and Module-Specific Commands), L2CAP, SDP, RFCOMM, TCS, BNEP, OBEX and TCI. Clicking a tab or a name in the protocol stack graphic opens a window and presents a menu of commands for that protocol.

5.1 Command Generator Interface



The Command Generator utility is composed of the following:

- **Tabs** — There are eight tabs: HCI, L2CAP, SDP, RFCOMM, TCS, BNEP, OBEX, and TCI. Clicking a tab displays the Command Menu for the chosen protocol.
- **Command Menus** — 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 will cause BTTrainer 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.
- **Command Description Box** — A short description will display in the Command Description Box when a command is selected from a Command Menu.
- **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.

5.2 Using Command Generator

Note: If Command Generator isn't enabled on your BTTrainer system, you will need to obtain a License Key from CATC before you can use it. See "License Keys" on page 16 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.

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.

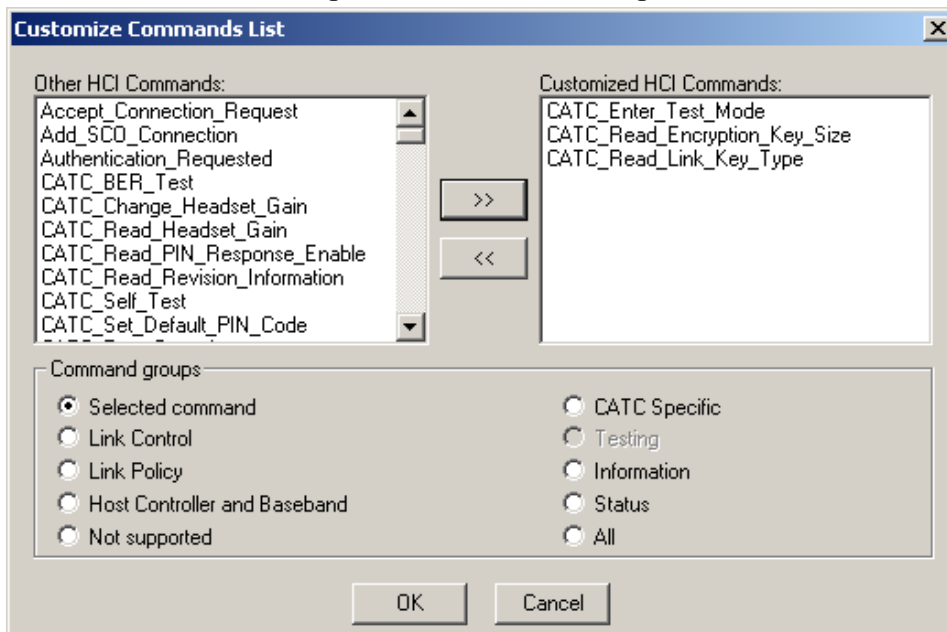
Step 5 Click the Execute button to run the command.

Note: While Command Generator offers maximum control over BTTrainer, 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 BTTrainer and a device, it is not possible to work with OBEX commands in Command Generator. BTTrainer 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.

5.2.1 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.



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.

5.3 Commands Available in Command Generator

The following tables summarize the commands in Command Generator.

For detailed descriptions of the commands, see Appendix A: Command Generator Command Descriptions, on page 83.

Note: “N/A” means Not Applicable. This indicates that the specified command does not have a parameter.

5.3.1 HCI Commands

Link Control Commands

Two sections of Link Control Commands are presented. The first section lists commands that are supported by BTTrainer. The second section presents commands that are not supported.

Table 5-1: Supported HCI Link Control commands

Commands	Parameters
Accept_Connection_Request	Request_Role_Switch
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 Packet_Type Page_Scan_Rep_Mode Page_Scan_Mode Clock_Offset Allow_Role_Switch
Disconnect	HCI_Handle
Exit_Periodic_Inquiry_Mode	N/A
GetSCOConnection	N/A
Inquiry	Inquiry_Length Num_Responses LAP
Inquiry_Cancel	N/A
Master_link_Key	Key_Flag
Periodic_Inquiry_Mode	Max_Period_Length Min_Period_Length Inquiry_Length Num_of_Responses LAP
PIN_Code_Request_Negative_Reply	BD_ADDR

Table 5-1: Supported HCI Link Control commands (Continued)

Commands	Parameters
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

Table 5-2: Unsupported HCI Link Control commands

Commands
Link_Key_Request_Negative_Reply
Link_Key_Request_Reply

Link Policy Commands

Table 5-3: Supported HCI Link Policy commands

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

Table 5-3: Supported HCI Link Policy commands (Continued)

Commands	Parameters
Switch_Role	BD_ADDR
Write_Link_Policy_Settings	HCI_Handle Link_Policy_Settings

Table 5-4: Unsupported HCI Link Policy commands

Commands
All Link Policy commands are supported in Command Generator.

Host Controller & Baseband Commands

Table 5-5: Supported HCI Host Controller & Baseband commands

Commands	Parameters
Change_Local_Name	Name
Create_New_Unit_Key	N/A
Delete_Stored_Link_Key	BD_ADDR Delete_All_Flag
Flush	HCI_Handle
Host_Buffer_Size	ACL_Data_Length SCO_Data_Length Total_Num_ACL Total_Num_SCO
Read_Authentication_Enable	N/A
Read_Automatic_Flush_Timeout	HCI_Handle
Read_Class_of_Device	N/A
Read_Connection_Accept_Timeout	N/A
Read_Current_IAC_LAP	N/A
Read_Encryption_Mode	N/A
Read_Hold_Mode_Activity	N/A
Read_Inquiry_Scan_Activity	N/A
Read_Link_Supervision_Timeout	HCI_Handle
Read_Local_Name	N/A
Read_Num_Broadcast_Retransmissions	N/A
Read_Number_Of_Supported_IAC	N/A
Read_Page_Scan_Activity	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_Transmit_Power_Level	HCI_Handle

Table 5-5: Supported HCI Host Controller & Baseband commands (Continued)

Commands	Parameters
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_Num_Broadcast_Retransmissions	Retransmissions
Write_Page_Activity	Page_Scan_Interval Page_Scan_Window
Write_Page_Scan_Mode	Page_Scan_Mode
Write_Page_Scan_Period_Mode	Page_Scan_Period_Mode
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

Table 5-6: Unsupported HCI Host Controller & Baseband commands

Commands
Host_Number_Of_Completed_Packets
Read_Transmit_Power_Level
Set_Host_Controller_To_Host_Flow_Control
Write_SCO_Flow_Control_Enable

Informational Commands

Table 5-7: Supported HCI Informational commands

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

Table 5-7: Supported HCI Informational commands (Continued)

Commands	Parameters
Read_Local_Supported_Features	N/A
Read_Local_Version_Information	N/A

Table 5-8: Unsupported HCI Informational commands

Commands
All Informational commands are supported in Command Generator.

Status Commands

Table 5-9: Supported HCI Status commands

Commands
Read_Failed_Contact_Counter
Reset_Failed_Contact_Counter

Table 5-10: Unsupported HCI Status commands

Commands
Get_Link_Quality
Read_RSSI

Testing Commands

Table 5-11: Supported HCI Testing commands

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

Note: The Supported HCI Testing commands shown above are part of the “TCI” dialog.

Table 5-12: Unsupported HCI Testing commands

Commands
All Testing commands are supported in Command Generator.

CATC-Specific Commands

Table 5-13: Supported CATC-Specific HCI commands

Commands	Parameters
CATC_Change_Headset_Gain	Device Gain
CATC_Decrease_Power_Request	HCI_Handle

Table 5-13: Supported CATC-Specific HCI commands (Continued)

Commands	Parameters
CATC_Disconnect	HCI_Handle TestMode Timeout
CATC_Get_ParkMode	N/A
CATC_Get_Selected_SCO_Connection	N/A
CATC_Increase_Power_Request	HCI_Handle
CATC_Max_Slot_Response	HCI_Handle Action Event
CATC_Modify_Beacon	HCI_Handle MaxBeaconInterval MinBeaconInterval
CATC_Max_Slot	HCI_Handle Max_Slot
CATC_Override_Remote_Features	HCI_Handle
CATC_Page_Mode_Request	HCI_Handle PageScheme PageSchemeSetting
CATC_Page_Scan_Mode_Request	HCI_Handle PageScheme PageSchemeSetting
CATC_Qos	HCI_Handle PollInterval Nbc
CATC_QoS_Response	Mode
CATC_Read_Encryption_Key_Size	N/A
CATC_Read_Headset_Gain	Device
CATC_Read_Link_Key_Type	N/A
CATC_Read_PIN_Response_Enable	N/A
CATC_Read_Revision_Information	N/A
CATC_Sco_Parameter_Change_Response	Mode
CATC_Select_Sco_Connection	HCI_Handle Data_Path Data
CATC_Self_Test	N/A
CATC_Set_Broadcast_Scan_Window	HCI_Handle BroadcastScanWindow
CATC_Set_Default_PIN_Code	Pin_Code
CATC_Set_Park_Mode	Park_Mode
CATC_Timing_Accuracy_Request	HCI_Handle

Table 5-13: Supported CATC-Specific HCI commands (Continued)

Commands	Parameters
CATC_Write_Encryption_Key_Size	Key_Size
CATC_Write_Link_Key_Type	Key_Type
CATC_Write_Local_Supported_Features	LMP_Features
CATC_Write_PIN_Response_Enable	PIN-Response_Enable

5.3.2 L2CAP Commands

Table 5-14: Supported L2CAP commands

Commands	Parameters
ConfigurationResponse	Reason TokenRate TokenBucketSize PeakBandWidth Latency DelayVariation
ConfigurationRequest	Reason TokenRate TokenBucketSize PeakBandWidth Latency DelayVariation
ConnectRequest	HCI_Handle PSM Receive MTU
ConnectResponse	Response
DeregisterPsm	PSM
DisconnectRequest	CID
EchoRequest	HCI_Handle Data
GroupRegister	PSM List of Bluetooth addresses
InfoRequest	HCI_Handle
RegisterPsm	PSM Receive MTU
SendData	CID Data Pipe

5.3.3 SDP Commands

Table 5-15: Supported SDP commands

Commands	Parameters
AddProfileServiceRecord	Profile ServerChannel
AddServiceRecord	Filename Record Name Server Channel
ProfileServiceSearch	HCI_Handle Profile
RequestServiceAttribute	HCI_Handle ServiceRecordHandle AttributeID AttributeID AttributeID
RequestServiceSearch	HCI_Handle ServiceClassID ServiceClassID ServiceClassID
RequestServiceSearchAttribute	HCI_Handle ServiceClassID ServiceClassID ServiceClassID
ResetDatabase	N/A

5.3.4 RFCOMM Commands

Table 5-16: Supported RFCOMM commands

Commands	Parameters
AcceptChannel	Accept
AcceptPortSettings	Accept
AdvanceCredit	(HCI/DLCI) Credit
CloseClientChannel	(HCI/DLCI)
CreditFlowEnabled	(HCI/DLCI)
DeregisterServerChannel	ServerChannel
OpenClientChannel	HCI_Handle ServerChannel MaxFrameSize Credit AttachTo
RegisterServerChannel	AttachTo
RequestPortSettings	(HCI/DLCI) BaudRate DataFormat FlowControl Xon Xoff

Table 5-16: Supported RFCOMM commands (Continued)

Commands	Parameters
RequestPortStatus	(HCI/DLCI)
SendATCommand	(HCI/DLCI) AT_Command
SendData	(HCI/DLCI) Data Pipe
SendTest	(HCI/DLCI)
SetLineStatus	(HCI/DLCI) LineStatus
SetModemStatus	(HCI/DLCI) ModemSignals Break Length

5.3.5 TCS Commands

Table 5-17: Supported TCS commands

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

5.3.6 OBEX Commands

Table 5-18: Supported OBEX commands

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

5.3.7 BNEP Commands in Command Generator

Table 5-19: BNEP Commands in Command Generator

Commands	Parameters
Accept	AcceptConnection
Open	BD_ADDR
Close	BNEP_ADDR
SetUpConnectionReq	BNEP_ADDR Destination UUID Source UUID
SendPkt	BNEP_ADDR
SendControlPkt	BNEP_ADDR Control Packet Type Range Start Range End
RegisterBNEP	N/A
DeregisterBNEP	N/A
SendPktGeneral	BNEP_ADDR data_to_send
SetControlTimeout	BNEP_ADDR Timeout

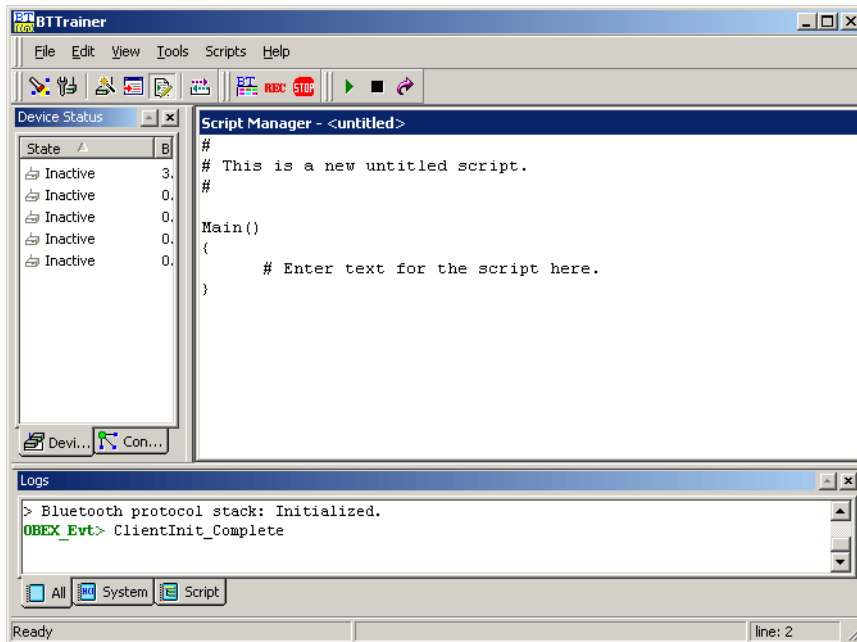
5.3.8 TCI Commands in Command Generator

Table 5-20: TCI Commands in Command Generator

Commands	Parameters
CATC_EnterTestMode	N/A
TestControlMaster	TestScenario Other parameters in SetErrors dialog
CATC_CreateConnectionExt	BD_ADDR Other parameters in SetErrors dialog
CATC_AcceptConnectionExt	TestScenario Other parameters in SetErrors dialog
CATC_SetClock	BluetoothClock
CATC_SetBdAddr	BD_ADDR
CATC_Page	BD_ADDR Number of Times TestScenario

Commands	Parameters
CATC_PageScan	TestScenario ClockOffset
CATC_Inquiry	TestScenario Number Of Times
CATC_InquiryScan	TestScenario ClockOffset Ninquiry
EnableDeviceUnderTestMode	N/A
ReadLoopbackMode	N/A
WriteLoopbackMode	LoopbackMode

CHAPTER 6: SCRIPT MANAGER



Script Manager is a tool within BTTrainer 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 BTTrainer. You can open a menu of these commands by clicking in the Script Manager window and then pressing <Control><Spacebar>. Command descriptions can be found in Appendix C: BTTrainer Scripting Commands.

6.1 Script Manager Interface

The Script Manager utility is composed of the following:

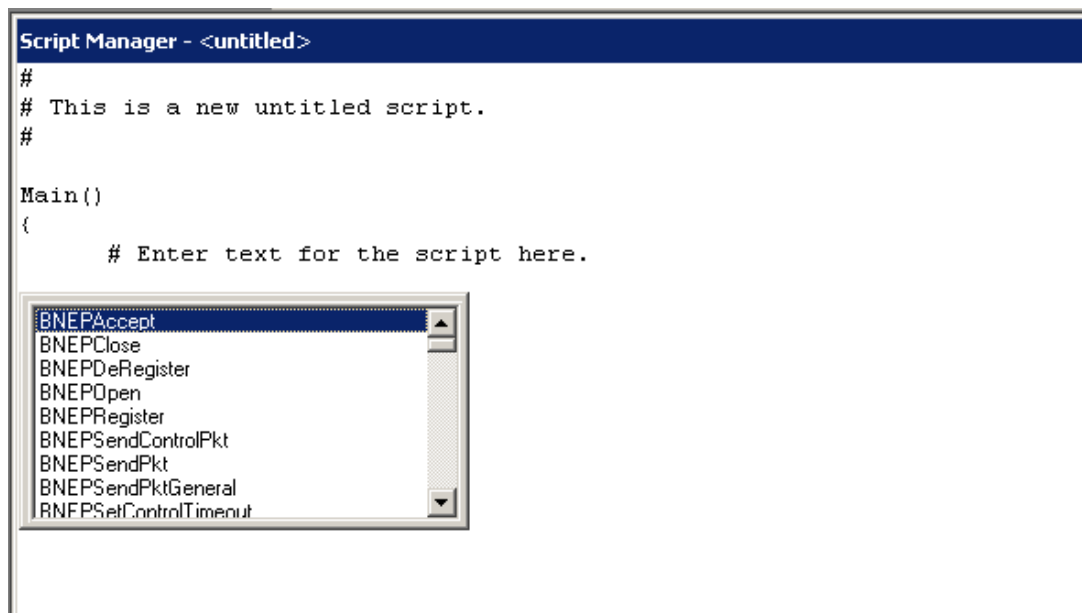
- **Work Area** -- The Work Area is a text editor for writing new scripts or displaying and editing opened scripts.
- **Run Button** -- Clicking the Run button saves (if needed) and executes the script that is currently displayed in the Work Area. While the script is running, the label on this button changes to Stop.
- **New Button** -- 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 Button** -- 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 Button -- Clicking the Save button saves the script that is currently open in the Work Area.
- Go To Button -- 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 BTTrainer application, on the status bar.
- Script Name and Path -- The name and path of the script that is currently open in the Work Area are displayed along the top of the Script Manager screen.
- 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.

6.1.1 Script Manager Pop-up Menu

As mentioned above, you can open a pop-up menu of commands by pressing <Control><Spacebar>. The pop-up menu provides you with a convenient means of remembering and entering commands supported by Script Manager.

To enter a command from the menu into the Script Manager window, select it, then press <Enter>.



6.2 Running Scripts

Note: If Script Manager isn't enabled on your BTTrainer system, you will need to obtain a License Key from CATC before you can use it. See "License Keys" on page 16 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.

Step 4 Click Run.

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.

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

6.3 Writing Scripts

Customized scripts can be written directly in Script Manager using BTTrainer Scripting Commands. This allows for automating sequences of commands. There are over 150 commands available for writing custom test sequences, including basic commands and commands for: pipes, HCI, L2CAP, SDP, RFCOMM, OBEX, and BTTracer. Detailed descriptions of the commands can be found in Appendix C: BTTrainer Scripting Commands, on page 147.

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

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.

Note: A convenient method of entering commands is to press <Control><Spacebar> while in the Script Manager window and select a command from the resulting pop-up menu.

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 BTTrainer Script file (*.script).

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

6.4 Sample Scripts

Sample scripts are provided with BTTrainer to demonstrate how Script Manager works. These scripts are subject to change and hence are not described here. Running sample.script will cause BTTrainer to attempt to connect to another device. The default location of the scripts is the directory where the application is installed, which is usually C:\Program Files\CATC\BTTrainer.

CHAPTER 7: 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.

The Device List in BTTracer shares the same data with the Device List in BTTrainer. Performing a Device Search in BTTracer will also update the Device List in BTTrainer.

Modifying existing device's data or adding new devices manually is possible through the BTTracer's list.

7.1 Device Search

BTTrainer 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.

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 BTTrainer 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.

BTTrainer 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.

7.2 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.

7.2.1 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 55.

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 BTTrainer has an ACL connection to the target device.

7.2.2 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 BTTrainer and a Bluetooth wireless audio device are successfully connected, a headset will need to be plugged into the audio port on BTTrainer. Be sure that the headset is plugged in before initializing the connection between BTTrainer 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 55.

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 BTTrainer has an ACL connection to the target device.

- 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 BTTrainer 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 BTTrainer has an SCO connection to the device.

- Step 5** To verify that BTTrainer and the Bluetooth device are successfully connected, speak into the microphone on one device and listen for audio on the other.

7.2.3 Display Device Information

Note: The following instructions assume that BTTrainer 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.

7.2.4 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.

To delete more than one device at a time, either

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

7.2.5 Disconnect All

A fast and easy way to terminate all connections that BTTrainer 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.

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 BTTrainer 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, BTTrainer 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.

CHAPTER 8: CONTACT AND WARRANTY INFORMATION

8.1 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

CATC Bluetooth Newsgroup

<http://bluenews.catc.com>

8.2 Limited Hardware Warranty

So long as you or your authorized representative ("you" or "your"), fully complete and return the registration card provided with the applicable hardware product or peripheral hardware products (each a "Product") within fifteen days of the date of receipt from Computer Access Technology Corporation ("CATC") or one of its authorized representatives, CATC warrants that the Product will be free from defects in materials and workmanship for a period of three years (the "Warranty Period"). You may also complete your registration form via the internet by visiting <http://www.catc.com/support/register/>. The Warranty Period commences on the earlier of the date of delivery by CATC of a Product to a common carrier for shipment to you or to CATC's authorized representative from whom you purchase the Product.

What this Warranty Does Not Cover

This warranty does not cover damage due to external causes including accident, damage during shipment after delivery to a common carrier by CATC, abuse, misuse, problems with electrical power, including power surges and outages, servicing not authorized by CATC, usage or operation not in accordance with Product instructions, failure to perform required preventive maintenance, software related problems (whether or not provided by CATC), problems caused by use of accessories, parts or components not supplied by CATC, Products that have been modified or altered by someone other than CATC, Products with missing or altered service tags or serial numbers, and Products for which CATC has not received payment in full.

Coverage During Warranty Period

During the Warranty Period, CATC or its authorized representatives will repair or replace Products, at CATC's sole discretion, covered under this limited warranty that are returned directly to CATC's facility or through CATC's authorized representatives.

How to Obtain Warranty Service

To request warranty service, you must complete and return the registration card or register via the internet within the fifteen day period described above and report your covered warranty claim by contacting CATC Technical Support or its authorized representative.

CATC Technical Support can be reached at 800-909-7112 or via email at support@catc.com. You may also refer to CATC's website at <http://www.catc.com> for more information on how to contact an authorized representative in your region. If warranty service is required, CATC or its authorized representative will issue a Return Material Authorization Number. You must ship the Product back to CATC or its authorized representative, in its original or equivalent packaging, prepay shipping charges, and insure the shipment or accept the risk of loss or damage during shipment. CATC must receive the Product prior to expiration of the Warranty Period for the repair(s) to be covered. CATC or its authorized representative will thereafter ship the repaired or replacement Product to you freight prepaid by CATC if you are located in the continental United States. Shipments made outside the continental United States will be sent freight collect.

Please remove any peripheral accessories or parts before you ship the Product. CATC does not accept liability for lost or damaged peripheral accessories, data or software.

CATC owns all parts removed from Products it repairs. CATC may use new and/or reconditioned parts, at its sole discretion, made by various manufacturers in performing warranty repairs. If CATC repairs or replaces a Product, the Warranty Period for the Product is not extended.

If CATC evaluates and determines there is "no trouble found" in any Product returned or that the returned Product is not eligible for warranty coverage, CATC will inform you of its determination. If you thereafter request CATC to repair the Product, such labor and service shall be performed under the terms and conditions of CATC's then current repair policy. If you chose not to have the Product repaired by CATC, you agree to pay CATC for the cost to return the Product to you and that CATC may require payment in advance of shipment.

General Provisions

THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE ADDITIONAL RIGHTS THAT VARY BY JURISDICTION. CATC'S RESPONSIBILITY FOR DEFECTS IN MATERIALS AND WORKMANSHIP IS LIMITED TO REPAIR AND REPLACEMENT AS SET FORTH IN THIS LIMITED WARRANTY STATEMENT. EXCEPT AS EXPRESSLY STATED IN THIS WARRANTY STATEMENT, CATC DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES FOR ANY PRODUCT INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF AND CONDITIONS OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTIES THAT MAY ARISE FROM ANY COURSE OF DEALING, COURSE OF PERFORMANCE OR TRADE USAGE. SOME JURISDICTIONS MAY NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE PRECEDING LIMITATION MAY NOT APPLY TO YOU.

CATC DOES NOT ACCEPT LIABILITY BEYOND THE REMEDIES SET FORTH IN THIS LIMITED WARRANTY STATEMENT OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES INCLUDING, WITHOUT LIMITATION, ANY LIABILITY FOR THIRD PARTY CLAIMS AGAINST YOU FOR DAMAGES, PRODUCTS NOT BEING AVAILABLE FOR USE, OR FOR LOST DATA OR SOFTWARE. CATC'S LIABILITY TO YOU MAY NOT EXCEED THE AMOUNT YOU PAID FOR THE PRODUCT THAT IS THE SUBJECT OF A CLAIM. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE PRECEDING EXCLUSION OR LIMITATION MAY NOT APPLY TO YOU.

The limited warranty on a Product may be transferred for the remaining term if the then current owner transfers ownership of the Product and notifies CATC of the transfer. You may notify CATC of the transfer by writing to Technical Support at Computer Access Technology Corporation, 2403 Walsh Avenue, Santa Clara, CA 95051-1302 USA or by email at: support@catc.com. Please include the transferring owner's name and address, the name and address of the new owner, the date of transfer, and the Product serial number.

APPENDIX A: COMMAND GENERATOR COMMAND DESCRIPTIONS

Note: “N/A” means Not Applicable. This indicates that the specified command does not have a parameter.

A.1 HCI Link Control Commands

A.1.1 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

A.1.2 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

A.1.3 Authentication_Requested

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

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

User's Manual

Return Events
Authentication_Error
Authentication_Complete

A.1.4 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

A.1.5 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	Range: 0x0000-0x0EFF 0x0008 = DM1 0x0010 = DH1 0x0400 = DM3 0x0800 = DH3 0x4000 = DM5 0x8000 = DH5

Return Events
Change_Connection_Packet_Type_Error
Change_Connection_Packet_Type_Complete

A.1.6 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.
Packet_Type		
Page_Scan_Rep_Mode		
Page_Scan_Mode		

User's Manual

Command Parameters	Examples	Comments
Clock_Offset		
Allow_Role_Switch		

Return Events
Create_Connection_Complete
Create_Connection_Error

A.1.7 Disconnect

Disconnect is used to terminate an existing connection.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

Return Events
Disconnection_Complete
Disconnection_Failed

A.1.8 Exit_Periodic_Inquiry_Mode

Exit_Periodic_Inquiry_Mode is used to end the Periodic Inquiry mode when BTTrainer is in Periodic Inquiry mode.

Command Parameters	Examples	Comments
N/A		

Return Events
Exit_Periodic_Inquiry_Mode_Complete
Exit_Periodic_Inquiry_Mode_Error

A.1.9 GetSCOConnections

Returns the handles of any active SCO connections to this device.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

User's Manual

Return Events
Success
Failure: Device not found
Failure

A.1.10 Inquiry

Inquiry will cause BTTrainer to enter Inquiry mode and discover other nearby Bluetooth devices.

Command Parameters	Examples	Comments
LAP		
Inquiry_Length	8	
Num_Responses	10	

Return Events
Inquiry_Complete
Inquiry_Result
Inquiry_Error

A.1.11 Inquiry_Cancel

Inquiry_Cancel will cause BTTrainer 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

A.1.12 Periodic_Inquiry_Mode

Periodic_Inquiry_Mode is used to configure BTTrainer 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

User's Manual

Command Parameters	Examples	Comments
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

A.1.13 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

A.1.14 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 (binary or ASCII)	1234	PIN_Code is a string character that can be up to 128 bits in length
BD_ADDR	0x010203040506	

Return Events
Command_Complete

User's Manual

A.1.15 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

A.1.16 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

A.1.17 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

A.1.18 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		

User's Manual

Return Events
Reject_Connection_Request_Complete

A.1.19 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

User's Manual

A.1.20 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

A.2 HCI Link Policy Commands

A.2.1 Get_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

A.2.2 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

A.2.3 Hold_Mode

Places the specified ACL link into Hold Mode.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

User's Manual

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

Return Events
Mode_Change
Hold_Mode_Error

A.2.4 Set_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

A.2.5 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

User's Manual

A.2.6 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

A.2.7 Role_Discovery

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

A.2.8 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

User's Manual**A.2.9 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

A.2.10 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

A.3 HCI Host Controller & Baseband Commands**A.3.1 Change_Local_Name**

Change_Local_Name allows the user-friendly name to be modified for the BTTrainer.

Command Parameters	Examples	Comments
Name	BTTrainer	Maximum string length =32 characters

Return Events
Change_Local_Name_Complete
Change_Local_Name_Error

User's Manual

A.3.2 Create_New_Unit_Key.

Command Parameters	Examples	Comments
NA		

A.3.3 Delete_Stored_Link_Key

Change_Local_Name allows the user-friendly name to be modified for the BTTrainer.

Command Parameters	Examples	Comments
Name	BTTrainer	Maximum string length =32 characters

Return Events
Change_Local_Name_Complete
Change_Local_Name_Error

A.3.4 Flush

Change_Local_Name allows the user-friendly name to be modified for the BTTrainer.

Command Parameters	Examples	Comments
HCI Handle		s

A.3.5 Host_Buffer_Size

Used by the BTTrainer to notify the BTTrainer Host Controller about its buffer sizes for ACL and SCO data. The BTTrainer Host Controller will segment the data to be transmitted from the Host Controller to BTTrainer, 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

User's Manual**A.3.6 Read_Authentication_Enable**

Read_Authentication_Enable will read the value for the Authentication_Enable parameter, which controls whether BTTrainer 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

A.3.7 Read_Automatic_Flush_Timeout

Command Parameters	Examples	Comments
HCI_Handle		

A.3.8 Read_Class_of_Device

Read_Class_of_Device will read the value for the Class_of_Device parameter for BTTrainer, 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

A.3.9 Read_Connection_Accept_Timeout

Read_Connection_Accept_Timeout will read the value for the Connection_Accept_Timeout parameter so that BTTrainer 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

User's Manual**A.3.10 Read_Current_IAC_LAP**

Read_Current_IAC_LAP will read the LAP(s) used to create the Inquiry Access Codes (IAC) that BTTrainer 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

A.3.11 Read_Encryption_Mode

Read_Encryption_Mode will read the value for the Encryption_Mode parameter, which controls whether BTTrainer 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

A.3.12 Read_Hold_Mode_Activity

Command Parameters	Examples	Comments
NA		

A.3.13 Read_Inquire_Scan_Activity

Command Parameters	Examples	Comments
NA		

A.3.14 Read_Link_Supervision_Timeout

Reads link supervision timeout setting for the specified ACL link.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

User's Manual

Return Events
Read_Link_Supervision_Timeout_Complete
Read_Link_Supervision_Timeout_Error

A.3.15 Read_Local_Name

Read_Local_Name reads the stored user-friendly name for BTTrainer.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Local_Name_Complete
Read_Local_Name_Error

A.3.16 Read_Num_Broadcast_Retransmission

Command Parameters	Examples	Comments
NA		

A.3.17 Read_Number_Of_Supported_IAC

This command will read the value for the number of Inquiry Access Codes (IAC) that BTTrainer 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

A.3.18 Read_Page_Scan_Activity

Command Parameters	Examples	Comments
N/A		

A.3.19 Read_Page_Scan_Mode

Read_Page_Scan_Mode command is used to read the current Page_Scan_Mode of BTTrainer.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Page_Scan_Mode_Complete Read_Page_Scan_Mode_Error

A.3.20 Read_Page_Scan_Period_Mode

Read_Page_Scan_Period_Mode is used to read the Page_Scan_Period_Mode of BTTrainer.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Page_Scan_Period_Mode_Complete Read_Page_Scan_Period_Mode_Error

A.3.21 Read_Page_Timeout

Read_Page_Timeout will read the value for the Page_Reply_Timeout configuration parameter, which allows BTTrainer 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

User's Manual**A.3.22 Read_PIN_Type**

Read_PIN_Type will read the PIN type specified in the host controller of BTTrainer.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_PIN_Type_Complete Read_PIN_Type_Error

A.3.23 Read_Scan_Enable

Read_Scan_Enable will read the value for the Scan_Enable configuration parameter, which controls whether or not BTTrainer 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

A.3.24 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, BTTracer can decide if the BTTrainer 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

User's Manual

A.3.25 Read_Transmit_Power_Level

Command Parameters	Examples	Comments
HCI Handle		
Type		

A.3.26 Read_Stored_Link_Key

Read_Stored_Link_Key will read one or all link keys stored in the BTTrainer 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

A.3.27 Read_Voice_Setting

Read_Voice_Setting will read the values for the Voice_Setting parameter in BTTrainer, 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

A.3.28 Reset

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

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

User's Manual

Return Events
Reset_Complete

A.3.29 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.
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

User's Manual

A.3.30 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

A.3.31 Write_Authentication_Enable

This command will write the value for the Authentication_Enable parameter, which controls whether BTTrainer 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

User's Manual

A.3.32 Write_Automatic_Flush_Timeout

Command Parameters	Examples	Comments
HCI_Handle		
Flush_Timeout		

A.3.33 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

A.3.34 Write_Connection_Accept_Timeout

Write_Connection_Accept_Timeout will write the value for the Connection_Accept_Timeout configuration parameter, which allows BTTrainer 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

User's Manual**A.3.35 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

A.3.36 Write_Encryption_Mode

Write_Encryption_Mode command will write the value for the Encryption_Mode parameter, which controls whether BTTrainer 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

User's Manual**A.3.37 Write_Hold_Mode_Activity**

Command Parameters	Examples	Comments
Activity		

A.3.38 Write_Inquiry_Scan_Activity

Command Parameters	Examples	Comments
Inquiry_Scan_Interval		
Inquiry_Scan_Window		

A.3.39 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

A.3.40 Write_Num_Broadcast_Retransmissions

Command Parameters	Examples	Comments
Retransmissions		

A.3.41 Write_Page_Scan_Activity

Command Parameters	Examples	Comments
Page_Scan_Interval		
Page_Scan_Window		

A.3.42 Write_Page_Scan_Mode

Command Parameters	Examples	Comments
Page_Scan_Mode		

A.3.43 Write_Page_Scan_Period_Mode

Command Parameters	Examples	Comments
Page_Scan_Period_Mode	0x0001	

A.3.44 Write_Page_Timeout

Write_Page_Timeout command will write the value for the Page_Reply_Timeout configuration parameter, which allows BTTrainer 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	<p>0=Illegal Page Timeout. Must be larger than 0</p> <p>N = 0xXXXX Page Timeout measured in Number of Baseband slots.</p> <p>Interval Length = N * 0.625 msec (1 Baseband slot)</p> <p>Range for N: 0x0001 – 0xFFFF</p> <p>Time Range: 0.625 msec -40.9 Seconds</p> <p>Default: N = 0x2000 Time = 5.12 Sec</p>

User's Manual

Return Events
Write_Page_Timeout_Complete
Write_Page_Timeout_Error

A.3.45 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

A.3.46 Write_Scan_Enable

The Write_Scan_Enable command will write the value for the Scan_Enable configuration parameter into BTTrainer, which controls whether or not BTTrainer 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

A.3.47 Write_Stored_Link_Key

Write_Stored_Link_Key command will write a link key to the BTTrainer host controller.

Command Parameters	Examples	Comments
BD_ADDR	0x010203040506	
Link_Key	0x01020304	

User's Manual

Return Events
Write_Stored_Link_Key_Complete
Write_Stored_Link_Key_Error

A.3.48 Write_Voice_Setting

The Write_Voice_Setting command will write the values for the Voice_Setting parameter into BTTrainer, 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

A.4 HCI Informational Commands**A.4.1 Read_BD_ADDR**

Read_BD_ADDR will read the value of BTTrainer'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

A.4.2 Read_Buffer_Size

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

Command Parameters	Examples	Comments
N/A		

User's Manual

Return Events
Read_Buffer_Size_Complete
Read_Buffer_Size_Error

A.4.3 Read_Country_Code

Read_Country_Code command will read the value for the Country_Code status parameter in BTTrainer. 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

A.4.4 Read_Local_Supported_Features

Read_Local_Supported_Features will request a list of the supported features for BTTrainer.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Local_Supported_Features_Complete
Read_Local_Supported_Features_Error

A.4.5 Read_Local_Version_Information

Read_Local_Version_Information command will read the values for the version information for BTTrainer.

Command Parameters	Examples	Comments
N/A		

Return Events
Read_Local_Version_Information_Complete
Read_Local_Version_Information_Error

A.5 Status Commands

A.5.1 Read_Failed_Contact_Counter

Command Parameters	Examples	Comments
HCI_Handle		

A.5.2 Reset_Failed_Contact_Counter

Command Parameters	Examples	Comments
HCI_Handle	0x0001	

A.6 HCI Testing Commands

A.6.1 Enable_Device_Under_Test_Mode

The Enable_Device_Under_Test_Mode command will allow BTTrainer to enter test mode via LMP test commands. BTTrainer 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

A.6.2 Read_Loopback_Mode

Read_Loopback_Mode will read the value for the setting of the BTTrainer 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

User's Manual**A.6.3 Write_Loopback_Mode**

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

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

A.7 CATC-Specific HCI Commands**A.7.1 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

A.7.2 CATC_Decrease_Power_Request

Send LMP_Decr_Power_Req PDU to the remote device.

Command Parameters	Examples	Comments
HCI_Handle		

Return Events
Command Status: Will return after the operation has started. Command Complete: Will return after the operation has completed.

User's Manual

Used in Test Cases

Test Case
TP/LIH/BV-35-C

A.7.3 CATC_Disconnect

This command will start a modified ACL disconnection procedure in order to verify that the IUT closes the link according to the TP/LIH/BV-04-C test case. During the procedure, LMP_Detach is sent, and after 3 seconds LMP_Features_Req PDUs is sent.

Command Parameters	Examples	Comments
HCI_Handle		
Test_Scenario		Values: 0 - Performs TP/PROT/ARQ/BV-25-C test case (Retransmissions of DV packet) 1 - Performs TP/PROT/ARQ/BV-26-C test case (Uncorrectable DV packet)
Timeout		In ms

Return Events
Command Complete: Will return after the operation has completed. Disconnection Complete event will return after the operation has completed: LMP_Detach and LMP_Features_Req are sent

Used in Test Cases

Test Case
TP/LIH/BV-04-C

A.7.4 CATC_Get_Park_Mode

This command gets current firmware mode that defines whether to park a device by PM_ADDR or by BD_ADDR when HCI_Park_Mode command is received..

Command Parameters	Examples	Comments
N/A		

Return Events
Command Complete: Will return after the operation has completed.

User's Manual

Used in Test Cases

Test Case	Description
TP/LIH/BV-26-C	Need to exit park mode by sending LMP_unpark_BD_ADDR_req.

A.7.5 CATC_Get_Selected_Sco_Connection

This command will return parameters for currently selected SCO connection.

Command Parameters	Examples	Comments
N/A		

Return Events
Command Complete: Will return after the operation has completed.

A.7.6 CATC_Increase_Power_Request

This command will send LMP_Incr_Power_Req PDU to the remote device predefined number of times.

Command Parameters	Examples	Comments
HCI_Handle		

Return Events
Command Status: Will return after the operation has started.
Command complete: Will return after the operation has completed.

Used in Test Cases

Test Case
TP/INF/BV-14-C
TP/LIH/BV-36-C

User's Manual

A.7.7 CATC_MaxSlot

This command will send LMP_Max_Slot PDU to the remote device.

Command Parameters	Examples	Comments
HCI_Handle MaxSlot		

Return Events
Command Status: Will return after the operation has started. Command complete: Will return after the operation has completed: LMP_Max_Slot PDU was sent.

Used in Test Cases

Test Case
TP/LIH/BV-64-C

A.7.8 CATC_MaxSlot_Response

This command will start a special Max Slot request/response procedure.

Command Parameters	Values	Description
HCI_Handle Action Event	0=Reject 1=Accept 0=EnableCateMaxSlot RequestEvent 1=DisableCateMaxSlot RequestEvent	Action=0/Event=0: Will reject all incoming LMP_Max_Slot_Req PDUs & generate HCI_CateMaxSlotRequestEvent for every PDU Action=1/Event=0: Will accept all incoming LMP_Max_Slot_req PDUs and generate HCI_CateMaxSlotRequestEvent for every PDU Action=1/Event=1: Will restore everything to the pre-script state

Return Events
Command Status: Will return after the operation has started. Command complete: Will return after the operation has completed: LMP_Max_Slot PDU was sent.

Used in Test Cases

Test Case
TP/LIH/BV-61-C TP/LIH/BV-64-C

User's Manual**A.7.9 CATC_Modify_Beacon**

This command will send LMP_Modify_Beacon PDU to notify parked devices about beacon parameter change.

Command Parameters	Examples	Comments
HCI_Handle		
BeaconMaxInterval		
BeaconMinInterval		

Return Events
Command Status: Will return after the operation has started.
Command complete: Will return after the operation has completed: LMP_Modify_Beacon PDU was sent.

Used in Test Cases

Test Case
TP/LIH/BV-25-C

A.7.10 CATC_Override_Remote_Features_Check

This command will override remote device LMP feature check in case local device needs to initiate a procedure that remote device doesn't support. Remote device LMP features are set in the LMP_Features_Req PDUs that are exchanged during the connection establishment. Example of the procedures are Sniff, Park, and Hold.

Command Parameters	Examples	Comments
HCI_Handle		

Return Events
Command complete: Will return after the operation has completed.

A.7.11 CATC_Page_Mode_Request

This command will send LMP_Page_Mode_Req PDU to the remote device.

Command Parameters	Examples	Comments
HCI_Handle		
PagingScheme		
PagingSchemeSettings		

User's Manual

Return Events
Command Status: Will return after the operation has started.
Command Complete event will return after the operation has completed: LMP_Page_Mode_Req PDU was sent

Used in Test Cases

Test Case	Description
TP/LIH/BV-65-C	Need to send LMP_Page_Mode_Req PDU
TP/LIH/BV-66-C	Need to send LMP_Page_Mode_Req PDU

A.7.12 CATC_Page_Scan_Mode_Request

This command will send LMP_Page_Scan_Mode_Req PDU to the remote device.

Command Parameters	Examples	Comments
HCI_Handle		.
PagingScheme		
PagingSchemeSettings		

Return Events
Command Status: Will return after the operation has started.
Command Complete event will return after the operation has completed: LMP_Page_Scan_Mode_Req PDU was sent

Used in Test Cases

Test Case	Description
TP/LIH/BV-70-C	Need to send LMP_Page_Scan_Mode_Req PDU
TP/LIH/BV-71-C	Need to send LMP_Page_Scan_Mode_Req PDU

A.7.13 CATC_Qos

This command will send LMP_Quality_Of_Service PDU over ACL connection according to the test case.

Command Parameters	Examples	Comments
HCI_Handle		
Poll_Interval		According to the poll_interval parameter of LMP_Quality_Of_Service PDU

User's Manual

Command Parameters	Examples	Comments
N_bc		According to the Nbc parameter of LMP_Quality_Of_Service PDU

Return Events
Command Status: Will return after the operation has started. Command Complete: Will return after the operation has completed.

Used in Test Cases

Test Case
TP/LIH/BV-39-C

A.0.1 CATC_QoS_Response

This command will configure the local device to accept or reject an incoming LMP_Quality_Of_Service_Request PDU for all connections.

Command Parameters	Examples	Comments
Mode		1 – Accept (default) 0 – Reject all incoming LMP_Quality_Of_Service PDU 2-0xFF reserved

Return Events
Command complete: Will return after the operation has completed.

Used in Test Cases

Test Case
TP/LIH/BV-41-C

A.7.14 CATC_Read_Encryption_Key_Size

Read the encryption key size.

Command Parameters	Examples	Comments
NA		

User's Manual

A.7.15 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

A.7.16 CATC_Read_Link_Key_Type

Reads Link Key Type.

Command Parameters	Examples	Comments
N/A		

A.7.17 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		

A.7.18 CATC_Read_Revision_Information

BTTrainer uses this command to read the revision number of the BTTrainer baseband controller.

Command Parameters	Examples	Comments
N/A		

Return Events
CATC_Read_Revision_Information_Complete
CATC_Read_Revision_Information_Error

A.7.19 CATC_Sco_Parameter_Change_Response

Configure local device to accept or reject an incoming request to change SCO parameters.

Command Parameters	Examples	Comments
Mode		

Return Events
Command Complete

Used in Test Cases

Test Case
TP/LIH/BV-56-C

A.7.20 CATC_Select_SCO_Connection

This command will select current SCO connection for up to three SCO connections. Only the incoming voice traffic (Rx Bluetooth packets) of the current SCO connection will be sent to the PCM CODEC. The outgoing voice traffic (Tx Bluetooth packets) of the current SCO connection is taken from the PCM CODEC or is set according to the "Data_Pattern" parameter. The source - PCM CODEC or "Data_Pattern" parameter - is selected by the "Data_Path" parameter. By default, the current SCO connection is the one that was established first. If the current SCO connection is disconnected and there is another one or two SCOs, the next current SCO connection is selected according to the following rules: if only one SCO connection remains, it is selected; if there are two SCO connections, one of

User's Manual

them is randomly selected to become the current connection. By default, "Data_Path" is set to "PCM", "Data_Length" is set to 0, and "Data_Pattern" is set to PRBS data. At least one SCO connection has to be open before calling this command.

Command Parameters	Values	Description
HCI_Handle		HCI Handle of the SCO connection to be selected.
Data_Path	0 - "PCM" (default) 1 - "HCI"	"PCM" - PCM CODEC is used to send and receive voice data for the current SCO connection. "HCI" - voice data is sent according to the "Data_Pattern" parameter. The received voice data is routed to the PCM CODEC.
Data	0-30	Valid only if "Data_Path=HCI" is selected.If 0, the data pattern is PRBS-9.
Data Path	Raw data pattern up to 30 bytes	Valid only if "Data_Path=HCI" is selected. If 0, the data pattern is PRBS-9. The pattern will be used for HV1, HV2, HV3, and both Voice and Data portions of DV packet. If Data_Length is less than the size of the voice packet, it'll be padded with '0'. If Data_Length exceeds the size of the voice packet, it'll be trimmed to fit into it.

Return Events
Command Complete: Will return after the operation has completed.

A.7.21 CATC_Self_Test

This command will perform self-test of BTTrainer.

Command Parameters	Examples	Comments
N/A		

Return Events
CATC_Self_Test_Complete
CATC_Self_Test_Error

User's Manual**A.7.22 CATC_Set_Broadcast_Scan_Window**

This command will send LMP_Set_Broadcast_Scan_Window PDU to the parked devices to set a new BroadcastScanWindow.

Command Parameters	Examples	Comments
HCI_Handle		
BroadcastScanWindow		Integer

Return Events
Generated Event: Command Status
Generated Event: Command Complete

A.7.23 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		

A.7.24 CATC_Set_Park_Mode

This command sets firmware mode that defines whether to park a device by PM_ADDR or by BD_ADDR when HCI_Park_Mode command is received. The default mode is parking by PM_ADDR. The command can be called at any time to specify which mode to use for parking devices. It will allow the selection of how individual devices will be parked.

Command Parameters	Examples	Comments
ParkMode		0 (default) - parking by PM_ADDR 1 - parking by BD_ADDR 2-0xFF - reserved

Return Events
Command Complete: Will return after the operation has completed.

Used in Test Cases

Test Case	Description
TP/LIH/BV-22-C	Need to park slave device using PM_ADDR=0

A.7.25 CATC_Timing_Accuracy_Request

This command will send LMP_Timing_Accuracy_Req PDU to the remote device.

Command Parameters	Examples	Comments
HCI_Handle		

Return Events
Command Status: Will return after the operation has started
Command Complete: Will return after the operation has completed.

Used in Test Cases

Test Case
TP/INF/BV-05-C

A.7.26 CATC_Write_Encryption_Key_Size

Write Encryption Key Size.

Command Parameters	Examples	Comments
Key_Size		

Return Events
Command Complete: Will return after the operation has completed.

A.7.27 CATC_Write_Link_Key_Type

This command write the Link Key Type.

Command Parameters	Examples	Comments
Key_Type		

Return Events
Command Complete: Will return after the operation has completed.

A.7.28 CATC_Write_Local_Supported_Features

This command will write new local LMP supported features parameters for the device. .

Command Parameters	Examples	Comments
LMP_Features		Bit Mask list of LMP features as defined in the Bluetooth 1.1 specification

Return Events
Command Complete: Will return after the operation has completed.

Used in Test Cases

Test Case
TP/PROT/PIC/BV-03-C
TP/PROT/PIC/BV-04-C

A.7.29 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.8 L2CAP Command Descriptions

A.8.1 ConfigurationResponse

Response to an incoming configuration request.

Command Parameters	Examples	Comments
Reason	“Accept”	“Accept” (Default) “Reject” “Reject - unacceptable params” “Reject - unknown options”

User's Manual

Command Parameters	Examples	Comments
TokenRate		
TokenBucketSize		
PeakBandwidth		
Latency		
DelayVariation		

Return Events
ConfigurationResponse_Complete

A.8.2 GroupRegister

Command Parameters	Examples	Comments
PSM		
BD_ADDR		

A.8.3 GroupDestroy

Command Parameters	Examples	Comments
CID		

A.8.4 GetRegisteredGroups

Command Parameters	Examples	Comments
NA		

A.8.5 ConfigurationSetup

Sets L2CAP connection options.

Command Parameters	Examples	Comments
ServiceType	0x01	
TokenRate	0x00	
TokenBucketSize	0x00	
PeakBandWidth	0x00	
Latency	0xFFFFFFFF	
DelayVariation	0xFFFFFFFF	

User's Manual

Return Events
ConfigurationSetup_Complete
Error

A.8.6 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

A.8.7 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

A.8.8 DeregisterPsm

Deregisters the specified PSM.

Command Parameters	Examples	Comments
PSM	0x1001	

Return Events
DeregisterPsm_Complete
DeregisterPsm_Failed

User's Manual

A.8.9 DisconnectRequest

Requests the disconnection of the specified L2CAP channel.

Command Parameters	Examples	Comments
CID	0x0040	

Return Events
Disconnection_Complete
Disconnection_Failed

A.8.10 EchoRequest

Sends an Echo Request over the L2CAP channel.

Command Parameters	Examples	Comments
HCI_Handle	0x0001	
Data	"echo"	

Return Events
EchoRequest_Complete
EchoRequest_Failed

A.8.11 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

A.8.12 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	

User's Manual

Command Parameters	Examples	Comments
Receive MTU	0x1B6	

Return Events
RegisterPsm_Complete RegisterPsm_Failed

A.8.13 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.9 Other L2CAP Events

Events
Connection_Indication
Disconnection_Indication
Data_Indication
Write_Configuration_Complete
Command_Complete
Error

A.10 SDP Command Descriptions

A.10.1 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: HeadsetAudioGateway Headset SerialPort Dialup Fax LAN FileTransfer ObjectPush Sync SyncCommand InterCom Cordless
ServerChannel	0x01	Server channel has to be entered for all profiles except for InterCom and Cordless.

Return Events
AddProfileServiceRecord_Complete
AddProfileServiceRecord_Error

User's Manual**A.10.2 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.
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

A.10.3 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: HeadsetAudioGateway Headset SerialPort Dialup Fax LAN FileTransfer ObjectPush Sync SyncCommand InterCom Cordless

Return Events
ProfileServiceSearch_Complete ProfileServiceSearch_Failed

User's Manual**A.10.4 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

A.10.5 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

User's Manual**A.10.6 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

A.10.7 ResetDatabase

This command will remove all Service Records from the database in BTTrainer.

Command Parameters	Examples	Comments
N/A		

Return Events
ResetDatabase_Complete
ResetDatabase_Failed

A.11 RFCOMM Command Descriptions**A.11.1 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)

User's Manual

Return Events
AcceptChannel_Complete

A.11.2 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

A.11.3 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

A.11.4 CloseClientChannel

This command will close an established RFCOMM channel between BTTrainer 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

User's Manual**A.11.5 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

A.11.6 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

A.11.7 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		

Return Events
OpenClientChannel_Complete
OpenClientChannel_Failed

User's Manual**A.11.8 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		

Return Events
RegisterServerChannel_Complete
RegisterServerChannel_Error

A.11.9 RequestPortSettings

This command will request a change to the current PortSettings.

Command Parameters	Examples	Comments
(HCI/DLCI)	(0x0001, 0x02)	
BaudRate	9600	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	0x02	The following values identify the number of data bits: 0x00 -- DATA_BITS_5 0x02 -- DATA_BITS_6 0x01 -- DATA_BITS_7 0x03 -- DATA_BITS_8
FlowControl	0x00	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	0x11	Default Xon char -- 0x11
Xoff	0x13	Default Xoff char -- 0x13

Return Events
RequestPortSettings_Complete
RequestPortSettings_Failed

User's Manual**A.11.10 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

A.11.11 SendData

Causes BTTrainer 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

A.11.12 SendTest

This command causes BTTrainer 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

A.11.13 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:

User's Manual

Command Parameters	Examples	Comments
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

A.11.14 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.
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

User's Manual**A.11.15 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.12 Other RFCOMM Events

Events
OpenClientChannel_Request
CloseClientChannel_Indication
Data_Indication
PortNegotiation_Indication
RequestPortStatus_Indication
ModemStatus_Indication
LineStatus_Indication
Flow_Off_Indication
Flow_On_Indication

A.13 TCS Command Descriptions

A.13.1 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

A.13.2 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

User's Manual**A.13.3 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

A.13.4 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

A.13.5 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.14 OBEX Command Descriptions

A.14.1 ClientAbort

Command Parameters	Examples	Comments
NA		.

A.14.2 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

A.14.3 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

A.14.4 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

User's Manual**A.14.5 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

A.14.6 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 a directory if it not exist. Returns an error instead.

Return Events
OBEX_Command_Complete ClientSetPath_Error

A.14.7 ServerDeinit

This command will deinitialize the OBEX server.

Command Parameters	Examples	Comments
N/A		

Return Events
ServerDeinit_Complete ServerDeinit_Error

User's Manual**A.14.8 ServerInit**

This command will initialize the OBEX server.

Command Parameters	Examples	Comments
N/A		

Return Events
ServerInit_Complete ServerInit_Error

A.14.9 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.15 BNEP Commands

A.15.1 Accept

Command Parameters	Examples	Comments
AcceptConnection		

A.15.2 Open

Command Parameters	Examples	Comments
BD_ADDR		.

A.15.3 Close

Command Parameters	Examples	Comments
BNEP_ADDR		

A.15.4 SetUpConnectionReq

Command Parameters	Examples	Comments
BNEP_ADDR		.
Destination UUID		
Source UUID		

A.15.5 SentPKT

Command Parameters	Examples	Comments
BNEP_ADDR		.
Packet_Type		

A.15.6 SendControlPKT

Command Parameters	Examples	Comments
BNEP_ADDR		
Control Packet Type		
Range Start		
Range End		

User's Manual

A.15.7 RegisterBNEP

Command Parameters	Examples	Comments
NA		

A.15.8 DeregisterBNEP

Command Parameters	Examples	Comments
NA		

A.15.9 SetControlTimeout

Command Parameters	Examples	Comments
BNEP_ADDR		
Timeout		

A.16 TCI Commands**A.16.1 CATC_EnterTestMode**

This command puts a remote slave device into test mode.

Command Parameters	Examples	Comments
N/A		

Return Events

Command Status: Will return after the operation has started.

Command Complete: Will return after the operation has completed.

A.16.2 CATC_TestControlMaster

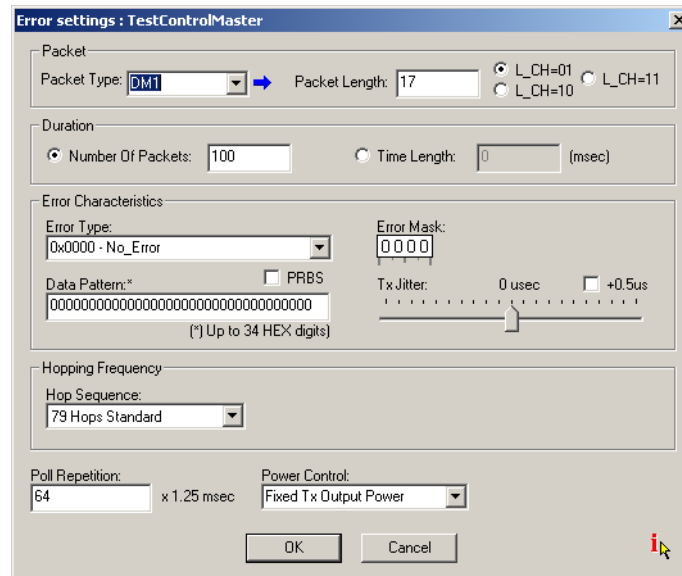
Enter Test Mode as a master device and start the test according to specified parameters.

Command Parameters	Examples	Comments
TestScenario		
Other Parameters in SetErrors dialog (see screenshot below)		

Return Events

Command Status: Will return after the operation has started.

Command Complete: Will return after the operation has completed.

User's Manual**Parameters**

Packet Type - Presents a menu of Packet types.

Packet Length - Enter a value in bytes.

L_CH=01, L_CH=10, L_CH=11 - Defines payload header L_CH value and type of payload to be sent. Data payload of L_CH=11b is restricted to a pseudo-random value that does not match any existing LMP PDU.

Duration - Set time in msec.

Time Length - Enter a value in msec.

Error Type - Presents a menu of error types.

Error Mask - Defines error distribution. Insert an error into a payload or in to header. This option is context sensitive and depends on the type of error selected

Example:

1. error_type = FEC 1/3 header, and error_mask=8000, 1-bit error is inserted into the header and the error bit will be the first bit in header.

2. error_type = FEC 2/3 payload, and error_mask=0c00, 2-bit error is inserted into the payload and the error bits will be the 5th and 6th bits in payload.

When running base band tests, make sure the mask starts from left (no leading zeros) since the FW will apply error mask to every bit depends on how many time the test will run.

PBRs - If checked, will generate random data.

Data Pattern - Define a pattern to be matched.

Tx Jitter - Random jitter range from -9.5 to 9.5. Check '+0.5us' box is 0.5 jitter as needed.

User's Manual

Hops Sequence - presents a menu with three options: 79 Hops Standard, Fixed Frequency, and Reduced Hop.

Poll Repetition - Presents a text box for entering a time value.

Power Control - Presents a menu with two options: Adaptive power control and Fixed Tx Output Power.

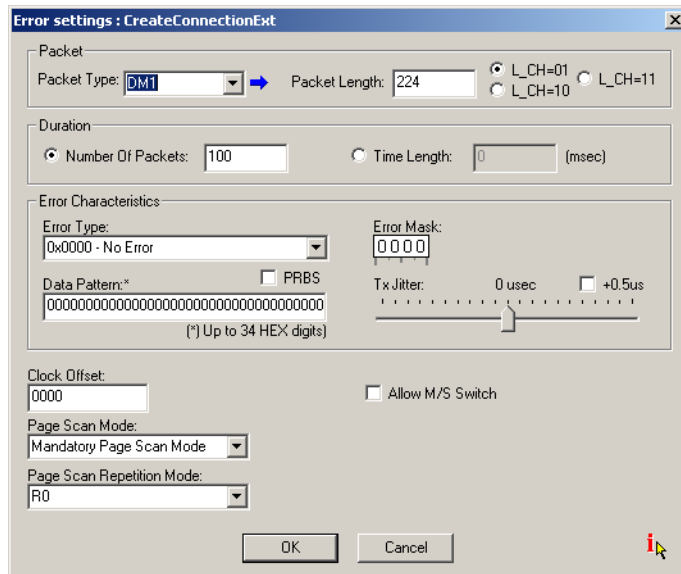
A.16.3 CATC_CreateConnectionExt

This command supports TCI functionality that requires BTTrainer to be connected as a master device to the IUT. This command will initiate a baseband connection process and perform the test defined by the parameters.

Command Parameters	Examples	Comments
BD_ADDR		
Other Parameters in SetErrors dialog		

Return Events

- Command Status: Will return after the operation has started.
- Command Complete: Will return after the operation has completed.



Parameters

Packet Type - Presents a menu of Packet types.

Packet Length - Enter a value in bytes.

User's Manual

L_CH=01, L_CH=10, L_CH=11 - Defines payload header L_CH value and type of payload to be sent. Data payload of L_CH=11b is restricted to a pseudo-random value that does not match any existing LMP PDU.

Duration - Set time in msec.

Time Length - Enter a value in msec.

Error Type - Presents a menu of error types.

Error Mask - Defines error distribution. Insert an error into a payload or in to header. This option is context sensitive and depends on the type of error selected

Example:

1. error_type = FEC 1/3 header, and error_mask=8000, 1-bit error is inserted into the header and the error bit will be the first bit in header.
2. error_type = FEC 2/3 payload, and error_mask=0c00, 2-bit error is inserted into the payload and the error bits will be the 5th and 6th bits in payload.

When running base band tests, make sure the mask starts from left (no leading zeros) since the FW will apply error mask to every bit depends on how many time the test will run.

PBRs - If checked, will generate random data.

Data Pattern - Define a pattern to be matched.

Tx Jitter - Random jitter range from -9.5 to 9.5. Check '+0.5us' box is 0.5 jitter as needed.

Hops Sequence - presents a menu with three options: 79 Hops Standard, Fixed Frequency, and Reduced Hop.

Poll Repetition - Presents a text box for entering a time value.

Power Control - Presents a menu with two options: Adaptive power control and Fixed Tx Output Power.

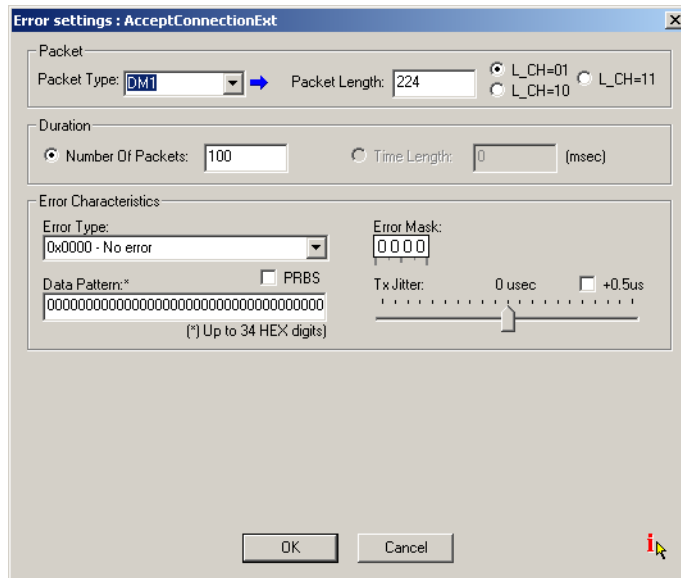
A.16.4 CATC_AcceptConnectionExt

This command supports TCI functionality that requires BTTrainer to be connected as a slave device. The command will accept an incoming connection request from the master, and configure BTTrainer to perform the specified test. This command can be executed several times outside the connection context to modify testing conditions.

Command Parameters	Examples	Comments
TestScenario		
Other Parameters in SetErrors dialog		

Return Events
Command Status: Will return after the operation has started.
Command Complete: Will return after the operation has completed.

User's Manual



Parameters

Packet Type - Presents a menu of Packet types.

Packet Length - Enter a value in bytes.

L_CH=01, L_CH=10, L_CH=11 - Defines payload header L_CH value and type of payload to be sent. Data payload of L_CH=11b is restricted to a pseudo-random value that does not match any existing LMP PDU.

Duration - Set time in msec.

Time Length - Enter a value in msec.

Error Type - Presents a menu of error types.

Error Mask - Defines error distribution. Insert an error into a payload or in to header. This option is context sensitive and depends on the type of error selected

Example:

1. error_type = FEC 1/3 header, and error_mask=8000, 1-bit error is inserted into the header and the error bit will be the first bit in header.

2. error_type = FEC 2/3 payload, and error_mask=0c00, 2-bit error is inserted into the payload and the error bits will be the 5th and 6th bits in payload.

When running base band tests, make sure the mask starts from left (no leading zeros) since the FW will apply error mask to every bit depends on how many time the test will run.

PBRs - If checked, will generate random data.

Data Pattern - Define a pattern to be matched.

Tx Jitter - Random jitter range from -9.5 to 9.5. Check '+0.5us' box is 0.5 jitter as needed.

User's Manual

Hops Sequence - presents a menu with three options: 79 Hops Standard, Fixed Frequency, and Reduced Hop.

Poll Repetition - Presents a text box for entering a time value.

Power Control - Presents a menu with two options: Adaptive power control and Fixed Tx Output Power.

A.16.5 CATC_SetClock

Command Parameters	Examples	Comments
BluetoothClock		

Return Events
Command Complete: Will return after the operation has completed.

A.16.6 CATC_SetBdAddr

Set the value of the local Bluetooth Device Address.

Command Parameters	Examples	Comments
BD_ADDR		

Return Events
Command Complete: Will return after the operation has completed.

A.16.7 CATC_Page

Initiate a modified Page procedure to the IUT.

Command Parameters	Examples	Comments
BD_ADDR		
TestScenario		
Number of Times		

Return Events
Command Status: Will return after the operation has started.
Command Complete: Will return after the operation has completed.

User's Manual

A.16.8 CATC_PageScan

Enter modified Page Scan mode.

Command Parameters	Examples	Comments
TestScenario		.
ClockOffset		

Return Events
Command Status: Will return after the operation has started.
Command Complete: Will return after the operation has completed.

A.16.9 CATC_Inquiry

Initiate a modified Inquiry procedure to the IUT.

Command Parameters	Examples	Comments
TestScenario		
Number of Times		

Return Events
Command Status: Will return after the operation has started.
Command Complete: Will return after the operation has completed.

A.16.10 CATC_InquiryScan

Enter modified Inquiry Scan mode.

Note: This command turns off the scan enable mode. You should call Write_Scan_Enable to restore the value.

Command Parameters	Examples	Comments
TestScenario		
ClockOffset		
Inquiry		

Return Events
Command Status: Will return after the operation has started.
Command Complete: Will return after the operation has completed.

Appendix B: *BTTrainer* Scripting Commands

BTTrainer 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.

B.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
}
```

B.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 BTTrainer

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 `GetRemoteDeviceName()`.

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 BTTrainer

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
```

B.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);
```

B.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!");
}

```

HCIAtcChangeHeadsetGain

HCIAtcChangeHeadsetGain(Device, Gain)

Parameter	Meaning	Default Value	Comments
Device	Speaker or microphone		Values: “Speaker”, “Microphone”
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");
    }
}

```

HCICatcDecreasePowerRequest

HCICatcDecreasePowerRequest (HCI_Handle)

Parameter	Meaning	Default Value	Comments
HCI_Handle			

Return value

- Command Status: Will return after the operation has started
- Command Complete: will return after the operation has completed.

Comments

This command will send LMP_Decr_Power_Req PDU to the remote device predefined number of times.

Example

```

Main()
{
    Device    = '008037163567';

    result = HCICatcIncreasePowerRequest(Device);
    Trace("HCICatcIncreasePowerRequest: ", result, "\n");

    result = HCICatcDecreasePowerRequest(Device);
    Trace("HCICatcDecreasePowerRequest: ", result, "\n");
    return result;
}

```

HCICatcDisconnect

HCICatcDisconnect (HCI_Handle, Mode, Timeout)

Parameter	Meaning	Default Value	Comments
HCI_Handle			
Mode		Values: 0-Tester Detach 1-IUT Detach 2-Link Supervision(Slave) 3- Link Supervision(Master)	
Timeout		Timeout value (in ms)	

Return value

- Command Complete event will return after the operation has started.
- Disconnection Complete event will return after the operation has completed: LMP_Detach and LMP_Features_Req are sent

Comments

This command will start a modified ACL disconnection procedure in order to verify that the IUT closes the link according to the TP/LIH/BV-04-C test case. During the procedure, LMP_Detach is sent, and after 3 seconds LMP_Features_Req PDUs is sent.

*Example***HCICatcGetParkMode**

HCICatcGetParkMode()

Parameter	Values	Description	Parameter
N/A			N/A

Return

- Command Complete event will return after the operation has completed: connection is placed in the active mode

Event Parameter	Values
Num_Hci_Command_Packets	
Command_Opcode	0xFC68
Status	0x00 – Success. Indicates successful start of the operation. 0x01 - Failure
ParkMode	0 - parking by PM_ADDR
1 Byte	1 - parking by BD_ADDR 2-0xFF - reserved

Used in Test Cases

Test Case	Description
TP/LIH/BV-26-C	Need to exit park mode by sending LMP_unpark_BD_ADDR_req.

HCICATCGetSelectedSCOConnection

HCICATCGetSelectedSCOConnection()

Parameter	Meaning	Default Value	Comments
N/A			

Return Events

- Command Complete: Will be returned after the operation has completed.

Comments

This command will return parameters for currently selected SCO connection.

Example

HCICATCIncreasePowerRequest

HCICATCIncreasePowerRequest (HCI_Handle)

Parameter	Meaning	Default Value	Comments
HCI_Handle			HCI_Handle
2 Bytes			2 Bytes

Return

- Command Status: Will return after the operation has started
- Command Complete: Will return after the operation has completed.

Used in Test Cases

Test Case
TP/INF/BV-14-C
TP/LIH/BV-36-C

Example

```

Main()
{
    Device      = '008037163567';

    result = HCICatcIncreasePowerRequest(Device);
    Trace("HCICatcIncreasePowerRequest: ", result, "\n");

    result = HCICatcDecreasePowerRequest(Device);
    Trace("HCICatcDecreasePowerRequest: ", result, "\n");

    return result;
}

```

HCICATCModifyBeacon

HCICATCModifyBeacon(HCI_Handle, BeaconMaxInterval, BeaconMinInterval)

Parameter	Meaning	Default Value	Comments
HCI_Handle			
2 Bytes			
BeaconMaxInterval			
2 Bytes			
BeaconMinInterval			
2 Bytes			

Return Events

- Command Status event will return after the operation has started
- Command Complete: Will return after the operation has been completed.

Comment

This command will send LMP_Modify_Beacon PDU to notify parked devices about beacon parameter change.

Used in Test Cases

Test Case
TP/LIH/BV-25-C

HCICATCMaxSlot

HCICATCMaxSlot (HCI_Handle, MaxSlot)

Parameter	Meaning	Default Value	Comments
HCI_Handle			
	2 Bytes		
MaxSlot			According to the LMP_MaxSlot PDU in the Bluetooth 1.1 HCI specification
	1 Byte		

Return Events

- Command Status event will return after the operation has started
- Command Complete: Will return after the operation has been completed.

Comment

Command Complete event will return after the operation has completed: LMP_Max_Slot PDU was sent.

Used in Test Cases

Test Case
TP/LIH/BV-64-C

Example

```

Main()
{
    Device    = '008037163567';
    MaxSlot   = 1;

    result = HCICatcMaxSlot(Device, MaxSlot);
    Trace("HCICatcMaxSlot: ", result, "\n");

    if(result != "Success")
    {
        return result;
    }

    MaxSlot = 3;
    result = HCICatcMaxSlot(Device, MaxSlot);
    Trace("HCICatcMaxSlot: ", result, "\n");
    if(result != "Success")
    {
        return result;
    }

    MaxSlot= 5;

```

User's Manual

```

    result = HCICatcMaxSlot(Device, MaxSlot);
    Trace("HCICatcMaxSlot: ", result, "\n");

    if(result != "Success")
    {
        return result;
    }

    return result;
}

```

HCICATCMaxSlotResponse

HCICATCMaxSlotResponse (HCI_Handle, Action, Event)

Parameter	Values	Description
HCI_Handle 2 Bytes		
Action 1 Byte	0-Reject 1-Accept	<p><i>Action=0/Event=0:</i> Will reject all incoming LMP_Max_Slot_Req PDUs and generate HCI_CatcMaxSlotRequestEvent for every PDU</p> <p><i>Action=1/Event=0:</i> Will accept all incoming LMP_Max_Slot_req PDUs and generate HCI_CatcMaxSlotRequestEvent for every PDU</p> <p><i>Action=1/Event=1:</i> Will restore everything to the pre-script state</p>
Event	0-EnableCatcMaxSlotRequestEvent 1-DisableCatcMaxSlotRequestEvent	

Return Events

- Command Status event will return after the operation has started
- Command Complete: Will return after the operation has been completed.

Comment

This command will start a special Max Slot request/response procedure for test case TP/LIH/BV-61-C and TP/LIH/BV-64-C.

Used in Test Cases

Test Case
TP/LIH/BV-61-C

TP/LIH/BV-64-C

HCICATCOVERRIDERemoteFeatureCheck

HCICATCOVERRIDERemoteFeatureCheck (HCI_Handle)

Parameter	Values	Description
HCI_Handle 2 Bytes		

Return Events

- Command Complete: Will return after the operation has been completed.

Comment

This command will override remote device LMP feature check in case local device needs to initiate a procedure that remote device doesn't support. Remote device LMP features are set in the LMP_Features_Req PDUs that are exchanged during the connection establishment. Example of the procedures are Sniff, Park, and Hold.

Example

```

Main()
{
    Device    = '008037163567';

    result = HCICatcOverrideRemoteFeaturesCheck(Device);
    Trace("HCICatcOverrideRemoteFeaturesCheck: ", result,
"\n");

    return result;
}

```

HCICATCPAGEModeRequest

HCICATCPAGEModeRequest (HCI_Handle, PagingScheme, PagingSchemeSettings)

Parameter	Values	Description
HCI_Handle 2 Bytes		
PagingScheme		According to the LMP_Page_Mode_Req PDU in the Bluetooth 1.1 HCI specification
PagingScheme-Settings		According to the LMP_Page_Mode_Req PDU in the Bluetooth 1.1 HCI specification

Return Events

- Command Status event will return after the operation has started
- Command Complete: Will return after the operation has been completed.

Comment

This command will send LMP_Page_Mode_Req PDU to the remote device

Used in Test Cases

Test Case
TP/LIH/BV-65-C
TP/LIH/BV-66-C

HCICATCPageScanModeRequest

HCICATCPageScanModeRequest (HCI_Handle, PagingScheme, PagingSchemeSettings)

Parameter	Values	Description
HCI_Handle 2 Bytes		
PagingScheme		According to the LMP_Page_Scan_Mode_Req PDU in the Bluetooth 1.1 HCI specification
PagingSchemeSettings		According to the LMP_Page_Scan_Mode_Req PDU in the Bluetooth 1.1 HCI specification

Return Events

- Command Status event will return after the operation has started
- Command Complete: Will return after the operation has been completed.

Comment

This command sends LMP_Page_Scan_Mode_Req PDU to the remote device.

Used in Test Cases

Test Case
TP/LIH/BV-70-C

TP/LIH/BV-71-C

HCICATCPageScanModeResponse

HCICATCPageScanModeResponse (Mode)

Parameter	Values	Description
Mode 1 Byte	0 -- Accept (default) 1 -- Reject 2-0xFF default	

Return Events

- Command Complete: Will return after the operation has been completed.

Comment

This command will configure local device to respond to an incoming LMP_Page_Scan_Mode_Req PDU.

Used in Test Cases

Test Case	Description
TP/LIH/BV-68-C	Need to reject an incoming send LMP_Page_Scan_Mode_Req PDU

HCICatcQos

HCICatcQos (HCI_Handle, Poll_Interval, N_bc)

Parameter	Values	Description
HCI_Handle 2 Bytes		
Poll_Interval 1 Byte		According to the poll_interval parameter of LMP_Quality_Of_Service PDU
Poll_Interval 1 Byte		According to the poll_interval parameter of LMP_Quality_Of_Service PDU

Return

- Command Status will return after operation has started
- Command Complete will return after the operation has completed

User's Manual*Comment:*

This command will send LMP_Quality_Of_Service PDU over ACL connection according to the test case.

Used in Test Cases

Test Case	Description
TP/LIH/BV-39-C	Generate LMP_quality_of_service PDU

HCICATCQoSResponse

HCICATCQoSResponse (Mode)

Parameter	Values	Description
Mode	1 -- Accept (default) 0 -- Reject all incoming LMP_Quality_Of_Service PDU 2 - 0xFF reserved	

Return Events

- Command Status event will return after the operation has started
- Command Complete will return after the operation has been completed.

Comment

Configure local device to accept or reject an incoming LMP_Quality_Of_Service_Request PDU

Used in Test Cases

Test Case
TP/LIH/BV-41-C

Example

```

Main()
{
    Mode = "Reject";
    result = HCICatcQoSResponse (Mode) ;
    Trace("HCICatcQoSResponse: ", result , "\n");

    Mode = "Accept";
    result = HCICatcQoSResponse (Mode) ;

```



```

        Trace("HCICatcQoSResponse: ", result , "\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.

HCICatcReadEncryptionKeySize

HCICatcReadEncryptionKeySize()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

Returns a list with two values: Status and KeySize

Status Return

- "Success"
- "Failure"

Comments

Read the size of the encryption key that is going to be used during an encryption process

Example

```
result = HCICatcReadEncryptionKeySize ();
Trace(result);
```

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 BTTrainer 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");
```

HCICatcScoErrorInjection

HCICatcScoErrorInjection(HCI_Handle, TestScenario)

Parameter	Meaning	Value	Comments
HCI_Handle 2 Bytes			
Test_Scenario 1 Byte		0, 1 2-0xFF - reserved	0- Performs TP/PROT/ARQ/BV-25-C test case (Retransmission of DV packet) 1- Performs TP/PROT/ARQ/BV-26-C test case (Uncorrectable DV packet)

Return

- Command Status will return after the operation has started.
- Command Complete will return after the operation has completed

Comment:

This command will send data over SCO connection with injected errors according to the test case.

HCICATCSCOParameterChangeResponse

HCICATCSCOParameterChangeResponse (Mode)

Parameter	Values	Description
Mode	1 -- Accept (default) 0 -- Reject all incoming parameter change 2 - 0xFF reserved	

Return Events

Command Complete will return after the operation has been completed.

Comment

This command will configure the local device to accept or reject an incoming request to change SCO parameters.

Used in Test Cases

Test Case	Description
TP/LIH/BV-56-C	Required to reject incoming SCO parameters change

Example

```

Main()
{
    Mode = "Reject";
    result = HCICatcScoParameterChangeResponse (Mode);
    Trace ("HCICatcScoParameterChangeResponse: ", result ,
"\n");

    Mode = "Accept";
    result = HCICatcScoParameterChangeResponse (Mode);
    Trace ("HCICatcScoParameterChangeResponse: ", result ,
"\n");
}

```

}

HCICATCSelectSCOConnection

HCICATCSelectSCOConnection(HCI_Handle, Data_Path, Data_Length)

Parameter	Meaning	Default Value	Comments
HCI_Handle			HCI Handle of the SCO connection to be selected.
Data_Path		0 - "PCM" (default) 1 - "HCI"	"PCM" -- PCM CODEC is used to send and receive voice data for the current SCO connection. "HCI" -- voice data is sent according to the "Data_Pattern" parameter. The received voice data is routed to the PCM CODEC.
Data_Length		0-30	Valid only if "Data_Path=HCI" is selected. If 0, the data pattern is PRBS-9.
Data_Pattern		Raw data pattern up to 30 bytes	Valid only if "Data_Path=HCI" is selected. If 0, the data pattern is PRBS-9. The pattern will be used for HV1, HV2, HV3, and both Voice and Data portions of DV packet. If Data_Length is less than the size of the voice packet, it'll be padded with '0'. If Data_Length exceeds the size of the voice packet, it'll be trimmed to fit into it.

Return Events

- Command Complete will return after the operation has completed.

Comments

Select current SCO connection among up to three SCO connections.

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.

User's Manual*Example*

```
Trace("Merlin's Wand Self Test : ", HCICatcSelfTest(),
"\n");
```

HCICATCSetBroadcastScanWindow

HCICATCSetBroadcastScanWindow(HCI_Handle, BroadcastScanWindow)

Parameter	Values	Description
HCI_Handle	2 Bytes	
BroadcastScanWindow	2 Bytes	According to the LMP_Set_Broadcast_Scan_Window PDU in the Bluetooth 1.1 HCI specification

Return Events

- Command Status event will return after the operation has started
- Command Complete: Will return after the operation has completed.

Comment

This command will send LMP_Set_Broadcast_Scan_Window PDU to the parked devices to set a new BroadcastScanWindow.

Used in Test Cases

Test Case
TP/LIH/BV-24-C

HCICatcSetDefaultPINCode

HCICatcSetDefaultPINCode (PINCode)

Parameter	Meaning	Default Value	Comments
PINCode			

Return value

- "Success"
- "Failure"
- "Invalid Parameter"

Comments

This command is used to write a default PIN code.

User's Manual*Example*

```
PINCode = "12345";
Result = HCICatcSetDefaultPINCode (PINCode)
Trace(Result);
```

HCICatcSetParkMode

HCICatcSetParkMode (ParkMode)

Parameter	Values	Description
ParkMode	0 (default) - parking by PM_ADDR	In the ParkMode=1, PM_ADDR is assigned 0.
1 Byte	1 - parking by BD_ADDR 2-0xFF - reserved	

Return

Command Complete event will return after the operation has completed

Used in Test Cases

Test Case	Description
TP/LIH/BV-22-C	Need to park slave device using PM_ADDR=0

HCICATCTimingAccuracyRequest

HCICATCTimingAccuracyRequest(HCI_Handle)

Parameter	Values	Description
HCI_Handle	2 Byte	

Return Events

- Command Status event will return after the operation has started
- Command Complete: Will return after the operation has completed.

Comment

This command will send LMP_Timing_Accuracy_Req PDU to the remote device

Used in Test Cases

Test Case	Description
TP/INF/BV-05-C	Need to send LMP_Timing_Accuracy_Req PDU

Example

```

Main()
{
    Device      = '008037163567';

    result = HCICatcTimingAccuracyRequest (Device);
    Trace("HCICatcTimingAccuracyRequest: ", result, "\n");
    if(result[0] == "Success")
    {
        Trace("Drift: ", result[1], " (ppm)\n");
        Trace("Jitter: ", result[2], " (ppm)\n");
    }
    return result;
}

```

HCICatcWriteEncryptionKeySize

HCICatcWriteEncryptionKeySize (KeySize)

Parameter	Meaning	Default Value	Comments
KeySize		Value can be from 1 to 16	

Return value

Returns a list with two values: status and KeySize

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Invalid Parameter"

Comments

Write the size of the encryption key that is going to be used during an encryption process

Example

```

KeySize = 7;

result = HCICatcWriteEncryptionKeySize (KeySize);
Trace(result);

```

HCICatcWriteLinkKeyType

HCICatcWriteLinkKeyType (KeyType)

Parameter	Meaning	Default Value	Comments
KeyType			Can be: "Combinational" or "Unit"

Return value

Returns a list with two values: status and KeyType

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Invalid Parameter"

Comments

Write the type of the link key that is going to be used during an authentication process

Example

```
KeyType = "Unit";

result = HCICatcWriteLinkKeyType (KeyType);
Trace(result);
```

HCICATCWriteLocalSupportedFeatures

HCICATCWriteLocalSupportedFeatures (LMP_Features)

Parameter	Meaning	Default Value	Comments
LMP_Features	Bit Mask list of LMP features as defined in Bluetooth spec		

Return value

- Generated Event Command Complete
- Command Complete will return after the operation has completed.

Comments

This command writes new local LMP supported features parameters for the device.

Used in Test Cases

Test Case	Description
TP/PROT/PIC/BV-03-C	Required to modify local LMP_Features to 0x0000000000000000
TP/PROT/PIC/BV-04-C	Required to modify local LMP_Features to 0x0000000000000000

Example

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

Main()
```



```

{
    result = HCIReadLocalSupportedFeatures();
    Trace("HCIReadLocalSupportedFeatures : ", result ,
"\n");

    features = '0102030400000000';
    result = HCICatcWriteLocalSupportedFeatures(features);
    Trace("HCICatcWriteLocalSupportedFeatures: ", result,
"\n");

    result = HCIReadLocalSupportedFeatures();
    Trace("HCIReadLocalSupportedFeatures : ", result ,
"\n");
}

```

HCICatcWritePinResponseEnable

HCICatcWritePinResponseEnable (PinResponseEnable)

Parameter	Meaning	Default Value	Comments
PinResponseEnable			Values are 0 (disabled) or 1 (enabled)

Return value

- "Success"
- "Failure"

Comments

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

Example

```

PinResponseEnable = 1;
result = HCICatcWritePinResponseEnable(PinResponseEnable);
Trace(result);

```

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!");
}

```

HCICreateNewUnitKey

HCICreateNewUnitKey ()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

This command is used to create a new unit key. The Bluetooth hardware will generate a random seed that will be used to generate the new unit key. All new connection will use the new unit key, but the old unit key will still be used for all current connections.

Example

```
result = HCICreateNewUnitKey();
Trace(result);
```

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
HCI_Handle	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");
```

HCIFlush

HCIFlush (Address)

Parameter	Meaning	Default Value	Comments
HCI_Handle			

Return value

- "Success"
- "Failure"
- "Not connected"
- "Failed: Not found"

Comments

The Flush command is used to discard all data that is currently pending for transmission in the Host Controller for the specified connection handle, even if there currently are chunks of data that belong to more than one L2CAP packet in the Host Controller.

Example

```
Device = '010203040506';
result = HCIFlush (Device);
Trace(result);
```

HCIGetSCOConnections

HCIGetSCOConnections (HCI_Handle)

Parameter	Meaning	Default Value	Comments
HCI_Handle			

Return value

- “Success”
- “Failure”
- “Failed: Device not found”

Comments

Returns the handles of any active SCO connections to this device.

Example

```
Trace("Device", Devices[0], " has these open SCO handles",
tail(GetSCOConnections(Devices[0])));
```

HCIIHoldMode

HCIIHoldMode (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");
}
```


User's Manual

```

        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”

User's Manual*Comments*

Specifies the PIN code to use for a connection.

Example

```
result = HCI PIN Code Request Reply('6C421742129F9', "New PIN
Code");
Trace("HCI PIN Code Request Reply 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		
Peak Bandwidth	The four-byte peak bandwidth value in bytes per second		
Latency	The four-byte latency value in microseconds		
Delay Variation	The four-byte delay variation 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 microseconds.

Comments

This command is used to specify Quality of Service parameters for the connection.

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");
```

HCIReadAutomaticFlushTimeout

HCIReadAutomaticFlushTimeout (Address)

Parameter	Meaning	Default Value	Comments
HCI_Handle			

Return value

Returns a list with two values: *status* and *FlushTimeout*

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Not connected"
- "Failed: Not found"

Comments

This command will read the value for the Flush_Timeout parameter for the specified connection handle.

Example

```
Address = '010203040506';
result = HCIReadAutomaticFlushTimeout (Address);
Trace(result);
```

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");
}

```

HCIReadFailedContactCounter

HCIReadFailedContactCounter (Address)

Parameter	Meaning	Default Value	Comments
Address			

Return value

Returns a list with two values: status and FailedContactCounter

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Not connected"
- "Failed: Not found"

Comments

This command will read the value for the Failed_Contact_Counter parameter for a particular connection to another device.

Example

```

Address = '010203040506';
result = HCIReadFailedContactCounter (Address);
Trace(result);

```

HCIReadHoldModeActivity

HCIReadHoldModeActivity ()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

Returns a list with two values: status and HoldModeActivity

Status (element 0) is one of the following:

- "Success"
- "Failure"

Comments

This command will read the value for the Hold_Mode_Activity parameter.

Example

```
result = HCIReadHoldModeActivity ();
Trace(result);
```

HCIReadInquiryScanActivity

HCIReadInquiryScanActivity()

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with three values: *status*, *InquiryScanInterval*, *InquiryScanWindow*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

InquiryScanInterval (element 1) and *InquiryScanWindow* (element 2) are the two byte values.

Comments

This command will read the value for *Inquiry_Scan_Activity* configuration parameter. The *Inquiry_Scan_Interval* configuration parameter defines the amount of time between consecutive inquiry scans. This is defined as the time interval from when the Host Controller started its last inquiry scan until it begins the next inquiry scan. The *Inquiry_Scan_Window* configuration parameter defines the amount of time for the duration of the inquiry scan. The *Inquiry_Scan_Window* can only be less than or equal to the *Inquiry_Scan_Interval*.

Example

```
result = HCIReadInquiryScanActivity();
Trace("HCIReadInquiryScanActivity returned: ", result[0],
"\n");
```

```

if (result[0] == "Success")
{
Trace("InquiryScanInterval is: 0x", result[1], "\n");
Trace("InquiryScanWindow is : 0x", result[2], "\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");
}
```

HCIReadNumBroadcastRetransmissions

HCIReadNumBroadcastRetransmissions()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

Returns Status and a value for the Number of Broadcast Retransmissions (Num_Broadcast_Retran)

Status is one of the following:

- "Success"
- "Failure"

Comments

This command will read the device's parameter value for the Number of Broadcast Retransmissions.

Example

```
result = HCIReadNumBroadcastRetransmissions ();
Trace(result);
```

HCIReadPageScanActivity

HCIReadPageScanActivity()

Parameter	Meaning	Default Value	Comments
N/A			

Return values

Returns a list with three values: *status*, *PageScanInterval*, *PageScanWindow*.

Status (element 0) is one of the following:

- "Success"
- "Failure"

PageScanInterval (element 1) and *PageScanWindow* (element 2) are the two byte values.

Comments

This command will read the value for *Page_Scan_Activity* configuration parameter. The *Page_Scan_Interval* configuration parameter defines the amount of time between consecutive page scans. This is defined as the time interval from when the Host Controller started its last page scan until it begins the next page scan.

The *Page_Scan_Window* configuration parameter defines the amount of time for the duration of the page scan. The *Page_Scan_Window* can only be less than or equal to the *Page_Scan_Interval*.

Example

```
result = HCIReadPageScanActivity();
Trace("HCIReadPageScanActivity returned: ", result[0],
"\n");
if (result[0] == "Success")
{
Trace("PageScanInterval is: 0x", result[1], "\n");
Trace("PageScanWindow is : 0x", result[2], "\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 sub-version 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 BTTrainer.

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		

Parameter	Meaning	Default Value	Comments
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");
}
```

HCIRjectConnectionRequest

HCIRjectConnectionRequest()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- “Success”

Comments

Sets the accept connection request variable to False.

Example

```
status = HCIRjectConnectionRequest();
Trace("HCIRjectConnectionRequest 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 ();
```

HCIResetFailedContactCounter

HCIResetFailedContactCounter (Address)

Parameter	Meaning	Default Value	Comments
Address			

Return value

- "Success"
- "Failure"
- "Not connected"
- "Failed: Not found"

Comments

This command will reset the value for the Failed_Contact_Counter parameter for a particular connection to another device.

Example

```
Address = '010203040506';
result = HCIResetFailedContactCounter (Address);
Trace(result);
```

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");
```

HCIWaitForMaxSlotRequest

HCIWaitForMaxSlotRequest (Address, Time)

Parameter	Meaning	Default Value	Comments
BD_ADDR		0	
	6 bytes		
Timeout		In ms	
	2 Bytes		

Comment:

This command will wait for event HCI_CatcMaxSlotRequestEvent or times out. This is a script only command.

Used in Test Cases

Test Case
TP/LIH/BV-61-C
TP/LIH/BV-64-C

HCIWriteAuthenticationEnable

HCIWriteAuthenticationEnable (AuthenticationEnable)

Parameter	Meaning	Default Value	Comments
AuthenticationEnable	Authentic- ation value: 0 = Authenti- cation dis- abled; 1 = Authenti- cation enabled for all connec- tions; 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);
```

HCIWriteAutomaticFlushTimeout

HCIWriteAutomaticFlushTimeout (Address, FlushTimeout)

Parameter	Meaning	Default Value	Comments
Address			

Parameter	Meaning	Default Value	Comments
FlushTimeout			

Return value

Returns a list with two values: status and FlushTimeout

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Not connected"
- "Failed: Not found"

Comments

This command will write the value for the Flush_Timeout parameter for the specified connection handle.

Example

```
Address = '010203040506';
FlushTimeout = 100;

result = HCIWriteAutomaticFlushTimeout
(Address, FlushTimeout);
Trace(result);
```

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
NumCurrentIACs	Number of current IACs.		Must be 1 or 2.
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 loopback 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);

```

HCIWritePageScanActivity

HCIWritePageScanActivity (PageScanInterval, PageScanWindow)

Parameter	Meaning	Default Value	Comments
PageScanInterval			
PageScanWindow			

Return value

Status is one of the following

- "Success"
- "Failure"

Comments

The Page_Scan_Interval configuration parameter defines the amount of time between consecutive page scans.

Example

```
PageScanInterval = 0x800;
PageScanWindow = 0x800;
Result =
HCIWritePageScanActivity(PageScanInterval, PageScanWindow);
Trace(Result);
```

HCIWritePageScanMode

HCIWritePageScanMode (PageScanMode)

Parameter	Meaning	Default Value	Comments
PageScanMode			

Return value

Returns a list with two values: status and PageScanMode

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Invalid Parameter"

Comments

Write the default page scan mode of the local device.

Example

```
PageScanMode = 0;
result = HCIWritePageScanMode (PageScanMode);
Trace(result);
```


HCIWritePageScanPeriodMode

HCIWritePageScanPeriodMode (PageScanPeriodMode)

Parameter	Meaning	Default Value	Comments
PageScanPeriodMode			

Return value

Returns a list with two values: Status and PageScanPeriodMode

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Invalid Parameter"

Comments

Write the mandatory Page_Scan_Period_Mode of the local device.

Example

```
PageScanPeriodMode = 0;
result = HCIWritePageScanPeriodMode (PageScanPeriodMode);
Trace(result);
```

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 BTTrainer.

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		

Parameter	Meaning	Default Value	Comments
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");
```

B.5 OBEX Commands

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"
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\\filename.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 (<i>not</i> relative) path and name of file to be sent from client.		

Parameter	Meaning	Default Value	Comments
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\\filename.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 "\\\".
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();
```

OBEXServerInit

OBEXServerInit()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- “Success”

- “Failure”

Comments

Initializes an OBEX server.

Example

```
result = OBEXServerInit();
```

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");
```

B.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");
```

RFRceiveData

RFRceiveData(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()
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 BTTrainer. 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");
}
```

RFSendATCommand

RFSendATCommand(Address, DLCI, AT_Command)

Parameter	Meaning	Default Value	Comments
Address			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()
AT_Command	AT command to send		ATCommand can be one of the following: "AT+CKPD=200", "RING", "OK", "ERROR", "AT+VGM=5", "AT+VGS=5", "+VGM=5", "+VGS=5", "CONNECT", "NO_CARRIER", "NO_DIALTONE", "BUSY"

Return value

- "Success"
- "Timed out"
- "Not supported" (invalid data type)
- "Not connected"

Comments

An RFCOMM connection must already be established with the device.

Example

```
Device = '010203040506 ' ;

result = RFSendATCommand(Device, dlci, "AT+CKPD=200");
```

```

        Trace(result);

    result = RFSendATCommand("CONNECTED_DEVICE", dlci, "RING");
    Trace(result);

```

RFDeregisterServerChannel

RFDeregisterServerChannel (ServerID)

Parameter	Meaning	Default Value	Comments
ServerID	ServerID of the channel that is to be deregistered	0	0 or 1

Return value

- "Success"
- "Failed: Not found"

Comments

Example

```

ServerID = 1;
result = RFDeregisterServerChannel(ServerID);
Trace("Result is ", result);

```

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"
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"

Parameter	Meaning	Default Value	Comments
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");
    }
}

```

RFSendTest ()

RFSendTest (Address, DLCI, Length, Pattern);

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to query		
DLCI	DLCI value to be used for the channel.		The DLCI value is used for the channel.
Length	Length of pattern in the command		
Pattern	Pattern to be sent		

Return value

Returns "Success", "Failure", "Not Connected", "Timed Out".

Comments

This command provides means to send an RFCOMM packet with the specified sequence of bytes.

Example

```
RFSendTest (Device, 2, 13, ['AB', 'CD', 'EF', '01', '23',
'45', '67', '89', 'AB', 'CD', 'EF', '01', '23']);
```

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");
}
```

RFSetCreditFlowControlEnable

RFSetCreditFlowControlEnable(enable);

Parameter	Meaning	Default Value	Comments
enable	Flag for enabling and disabling the Credit Flow Control		BNEP connection has to be present.

Return value

Always returns "Success"

Comments

By default BTTrainer uses RFCOMM Credit Flow Control.

If enable = 0- Credit Flow Control will be disabled for all new RFCOMM channels opened by RFRegisterServerChannel() and RFOpenClientChannel() commands (device will work as 1.0B compliant device).

If enable != 0- Credit Flow Control will be enabled for all new RFCOMM channels opened by RFRegisterServerChannel() and RFOpenClientChannel() commands (device will work as 1.1 compliant device).

Example

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");
}
```

B.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 Num- ber		

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");
}

```

B.8 BNEP Commands

BNEPAccept ()

BNEPAccept (Status)

Parameter	Meaning	Default Value	Comments
Status	"Yes" or "No"	"No"	

Return value

- "Success"
- "Failure"

Comments

Defines whether to accept or reject an incoming BNEP connection.

Example

```
Result = BNEPAccept ("Yes");
```

BNEPClose ()

BNEPClose (BNEP_Address)

Parameter	Meaning	Default Value	Comments
BNEP_Address	BNEP Address of the connection to be closed		

Return value

- "Success"
- "Failure"
- "Failed: Not found"
- "No BNEP connection"
- "Not connected"

Comments

Close BNEP connection with BNEP_Address.

Example

```
BNEP_Address = '010203040506';
Result = BNEPClose(BNEP_Address);
```

BNEPDeregister

BNEPDeregister ()

Parameter	Meaning	Default Value	Comments
-----------	---------	---------------	----------

Return value

- "Success"
- "Failure"

Comments

Deregister BNEP protocol.

Example

```
Result = BNEPDeregister();
Trace(Result);
```


BNEPOpen ()

BNEPOpen (Address)

Parameter	Meaning	Default Value	Comments
Address	Address of the remote Blue-tooth device		

Return value

Returns a list with the following values: status , BNEP_Address, ChannelID, ControlTimeout.

Status (element 0) is one of the following:

- "Success"
- "Failure"
- "Not connected"
- "Failed: Not found"
- "Timed out"

Comments

Open BNEP channel to a remote device. ACL connection to that device must exist and BNEP has to be registered.

Example

```
Address = '010203040506';
Result = BNEPOpen(Address);
Trace(Result);
```

BNEPRegister ()

BNEPRegister ()

Parameter	Meaning	Default Value	Comments
None			

Return value

- "Success"
- "Failure"

Comment

Register BNEP protocol. Call this command before using any other BNEP command.

Example

```
Result = BNEPRegister();
Trace(Result);
```

BNEPSendControlPkt()

BNEPSendControlPkt(BNEP_ADDR, packet type)

Parameter	Meaning	Default Value	Comments
BNEP_Address			
Packet_Type	Possible types are: "filter net type set msg" "filter multi addr set msg" "unknown control packet"		

Return value

- "Success"
- "Failure"
- "Invalid Parameter"
- "Timed out"
- "Failed: Not found"
- "No BNEP connection"
- "Not connected"

Comments

Send a predefined BNEP control packet to the remote device.

Example

```
BNEP_ADDR = '010203040506';
packet_type = "filter net type set msg";
Result = BNEPSendControlPkt(BNEP_ADDR, packet type);
```

BNEPSendPkt()

BNEPSendPkt(BNEP_ADDR, packet_type)

Parameter	Meaning	Default Value	Comments
BNEP_Address			

Parameter	Meaning	Default Value	Comments
Packet_Type	Possible types are: "compressed ethernet" "comp ethernet source only" "comp ethernet dest only" "general ethernet" "general with ext hdr"		

Return value

- "Success"
- "Failure"
- "Invalid Parameter"
- "Timed out"
- "Failed: Not found"
- "No BNEP connection"
- "Not connected"

Comments

Send a predefined BNEP packet to the remote device.

Example

```
BNEP_ADDR = '010203040506';
packet_type = "compressed ethernet";

Result = BNEPSendPkt(BNEP_ADDR,packet_type);
Trace(Result);
```

BNEPSendPktGeneral()

BNEPSendPktGeneral(BNEP_ADDR, data_to_send)

Parameter	Meaning	Default Value	Comments
BNEP_Address	BNEP address of the remote device.		BNEP connection has to be present.

Parameter	Meaning	Default Value	Comments
data_to_send	byte stream which will be sent over this connection as a BNEP command		

Return value

- "Success"
- "Timeout"
- "Invalid Parameter"
- "Busy"
- "Failure"

Comments

This command provides means to send a sequence of bytes over a connection as a BNEP command.

*Example**Example*

```
Result = BNEPSetControlTimeout(BNEP_Address, Timeout)
Trace(Result);BNEPSetUpConnectionReq()
```

BNEPSetUpConnectionReq()

BNEPSetUpConnectionReq(BNEP_ADDR, Destination_UUID, Source_UUID)

Parameter	Meaning	Default Value	Comments
BNEP_ADDR			
Destination_UUID	Possible values: "PANU" "GN" "NAP"		
Source_UUID	Possible values: "PANU" "GN""NAP"		

Return value

- "Success"

- Failure"
- Invalid Parameter"

Comments

Send connection request to remote device to inform device of SDP service UUIDs

Example

```
BNEP_ADDR = '010203040506';
Destination_UUID = "NAP";
Source_UUID = "GN";
Result =
BNEPSetUpConnectionReq(BNEP_ADDR, Destination_UUID, Source_U
UID)
```

BNEPSetControlTimeout()

BNEPSetControlTimeout(BNEP_Address, Timeout)

Parameter	Meaning	Default Value	Comments
BNEP_Address			
Timeout			The timeout value within 1 and 30

Return value

- Success"
- Failure"
- "Invalid Parameter"

Comments

Set a timeout for BNEP control commands.

Example

```
Result = BNEPSetControlTimeout(BNEP_Address, Timeout)
Trace(Result);
```

B.9 L2CAP Commands

L2CAPConfigurationRequest

L2CAPConfigurationRequest(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.
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.

Parameter	Meaning	Default Value	Comments
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”

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");
}

```

L2CAPGroupRegister

L2CAPGroupRegister(PSM, list of Bluetooth addresses)

Parameter	Meaning	Default Value	Comments
PSM	PSM of the group		
List of the Bluetooth addresses			

Return value

Returns a list with two values: status and GroupID

Status (element 0) is one of the following:

- "Success"
- "Failed"
- "Invalid Parameter"

Comments

Register an L2CAP group with the specified PSM and the list of Bluetooth addresses.

Example

```

PSM = 0x1001;
Addr1 = '010203040506';
Addr2 = '010203040507';
result = L2CAPGroupRegister (PSM, Addr1, Addr2);
Trace(result);

```

L2CAPGetRegisteredGroups

L2CAPGetRegisteredGroups()

Parameter	Meaning	Default Value	Comments
Address			

Return value

Returns a list with the following values: [status, GroupList]

Status (element 0) is one of the following:

- ·"Success"
- ·"Timed out"
- ·"Failed: Not found"

GroupList has a format [Group1...GroupN]

Group has a format [GroupCID, GroupPSM, GroupMember1...GroupMemberM]

Comments

Returns the list of all registered L2CAP groups for connectionless L2CAP connections.

Example

```
result = L2CAPGetRegisteredGroups ();
Trace(result);
```

L2CAPGroupDestroy

L2CAPGroupDestroy (GroupID)

Parameter	Meaning	Default Value	Comments
GroupID	An ID of the group that is to be destroyed.		

Return value

- "Success"
- "Failed"

Comments

Destroys an L2CAP group with the specified GroupID.

Example

```
GroupID = 0x0001;
result = L2CAPGroupDestroy (GroupID);
Trace(result);
```

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			

Parameter	Meaning	Default Value	Comments
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
}
```

B.10 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 BTTrainer SDP database

Example

```
SDPAddProfileServiceRecord(rfChannel, "ObjectPush");
```


SDPAddServiceRecord

SDPAddServiceRecord(FileName, RecordName, ServerChannel)

Parameter	Meaning	Default Value	Comments
FileName	String containing the full path of the file that contains the record		
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");
```

SDPGenericQuery

SDPGenericQuery(Address, PduID, size, data_to_send)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to query		

User's Manual

Parameter	Meaning	Default Value	Comments
PduID	PDU ID that would be used for the request	1	1 byte value
Size	Number of bytes to be used from 'data.'		
data_to_send	Byte values to be used for constructing the ASDP request.		Hex values

Return value

- "Success"
- "Failure"

Comments

This command provides means to send an SDP request with a sequence of bytes provided by the user. Allows the customization of SDP request to send requests with invalid parameters.

The SDP request would be composed of the following fields:

PDU ID - as specified by the PduID parameter.

Transaction ID - Calculated internally by the system

Length - Calculated internally by the system.

Parameters - The data following the 5th byte is taken from the 'data_to_send' filed in the command.

Example

```
result = SDPGenericQuery(Device,
    0xB8, 8,
    'DD330BB01ABEF0123456789ABCDEF0123456789ABCDEF11111');
if (result != "Success")
{
    Trace("SDPGenericQuery() - Failed. Result=", result,
    "\n");
}
```

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", "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);
}
```

SDPRequestServiceSearch

SDPRequestServiceSearch (Address, ParamsNum, ServiceID)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to query		
ParamsNum	Number of parameters	1	1-3
ServiceID	Service IDs to search		Hex

Return value

- SDP Services
- "Failure "

Comments

Locate service records based on a search pattern.

Example

```
result = SDPRequestServiceSearch(Device[0], 2, 0x1000,
0x1001, 0);
if (result != "Success")
{
    Trace("SDPRequestServiceSearch() - Failed. Result=",
result, \n");
}
```

SDPRequestServiceAttribute

SDPRequestServiceAttribute(Address, ServiceRecordHandle, AttributesNum, Attribute)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to query		
ServiceRecordHandle	Handle of the record that holds the attributes		
AttributesNum	Number of attributes	1	1-3
Attribute1-Attribute3	Attribute values to search		

Return value

- SDP Attribute values
- "Failure"

Comments

Retrieve specified attribute values from a specified service record.

Example

```
result = SDPRequestServiceAttribute(Device, 0, 2, 0x0001,
0x0002, 0);
if (result != "Success")
{
    Trace("SDPRequestServiceAttribute() - Failed. Res=",
result, "\n");
}
```

SDPRequestServiceSearchAttribute

SDPRequestServiceSearchAttribute (Address, AttributesNum, Attribute)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of device to query		
AttributesNum	Number of attributes	1	1-3
Attribute1-Attribute3	Attribute values to search		

Return value

- SDP Attribute values
- "Failure"

Comments

Retrieve attribute values that match the service search pattern.

Example

```
result = SDPRequestServiceSearchAttribute(Device, 1,
0x3234, 0, 0);
if (result != "Success")
{
    Trace("SDPRequestServiceSearchAttribute() - Failed.
Res=", result, "\n");
}
```

SDPResetDatabase

SDPResetDatabase ()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- "Success"
- "Failure"

Comments

Clears all records out of the BTTrainer SDP profile database.

Example

```
SDPResetDatabase ();
```

B.11 TCI Commands

HCICatcAcceptConnectionExt

HCICatcAcceptConnectionExt (TestScenario, PacketType, NumberOfPackets, Data, ErrorType, ErrorMask, Tx Jitter)

Comments

This command supports TCI functionality that requires BTTrainer to be connected as a slave device. The command will accept an incoming connection request from the master, and configure BTTrainer to perform the specified test.

This command can be executed several times outside the connection context to modify testing conditions.

Parameter	Meaning	Default Value	Comments
TestScenario			Values: 0 - Comply with the regular Bluetooth connection 1- Send packets in every slave Tx slot
PacketType			Values defined for HCI_CreateConnection command
NumberOfPackets			Values: 0x0001 -- 0xFFFF
Data			
ErrorType			See the table below

User's Manual

Parameter	Meaning	Default Value	Comments
ErrorMask			<p>0 - Random error distribution >0 - Error bit positions</p> <p>If ErrorType is not zero, this parameter is valid. It defines error distribution. For uncorrectable FEC 2/3, random error distribution, several erroneous bits will be inserted in every 15-bit packet. Valid for DM packet types For uncorrectable FEC 1/3 header, random error distribution, two error bit will be inserted in every 3-bit packet. Valid for all packet types with header.</p> <p>Random jitter between --10 and +10us is added to the nominal 625 us Slave Rx/Tx timing. The value is 2's compliment. Example of Jitter values are: 0 -- 0x00 1 -- 0x01+5 -- 0x05-1 -- 0xFF-5 - 0xFB+10 - 0x0A-10 - 0xF6</p>
Tx Jitter			<p>Random jitter between -10 and +10us is added to the nominal 1250 us Master Tx timing. The value is 2's compliment. 0.5us is supported. For example, 64 wil give a jitter of 0.5us, 70 will give jitter of 5.5us.</p> <p>Example of Jitter values are: 0-10 + Jitter 0 to10, w/o 0.5us 64-74+ Jitter 0 to10, w/ 0.5us -64 to -74- Jitter -1 to -10, w/ 0.5us- -1 to -10 - Jitter -1 to -10, w/o 0.5us</p>
L_CH	<p>Defines payload header L_CH value and type of the payload to be sent</p> <p>1 2 3 0 and 4 - 0xFF</p>		<p>Values: L_CH=2b'01 Continuation fragment of an L2CAP message (Default value) L_CH=2b'10 Start of an L2CAP message or no fragmentation L_CH = 2'b11 LMP Message Reserved</p>

User's Manual*ErrorType*

0x0000	No error
0x0001	Send NULL packets with NAK as the response to the incoming data packets
0x0002	Implicit NAK. No response to the incoming data packets.
0x0004	Uncorrectable FEC 1/3 header error injection
0x0008	Uncorrectable FEC 2/3 payload error injection
0x0010	FLOW control bit set to 0. NAK bit is set as well.
0x0020	FLOW control bit set to 0. NAK bit is set as well. Implicit GO.
0x0040	Same SEQN value
0x0080	Wrong UAP
0x0100	Deadlock avoidance according to the LMP test case TP/LIH/BV-81-C

Return value

- "Success"
- "Failure"
- "Invalid Parameter"

Used in Test Cases

Test Case
TP/PHYS/TRX/BV-04-C
TP/PROT/ARQ/BV-14-C
TP/PROT/ARQ/BV-15-C
TP/PROT/ARQ/BV-16-C
TP/PROT/ARQ/BV-18-C
TP/PROT/ARQ/BV-19-C
TP/PROT/ARQ/BV-23-C
TP/PROT/CON/BV-08-C

Example

```

#
# HCI_CatcAcceptConnectionExt
#

Main()
{
    TestScenario           = 1;

```

```

PacketType          = ["DM3"];
NumOfPackets        = 100;
Data                = 0x01020304;
ErrorType           = 0x0000;
ErrorMask           = 0x0000;
TxJitter            = 0x00;
L_CH                = 2;

result = HCICatcAcceptConnectionExt (TestScenario,
                                     PacketType,
                                     NumOfPackets,
                                     Data,
                                     ErrorType,
                                     ErrorMask,
                                     TxJitter,
                                     L_CH );

Trace("HCICatcAcceptConnectionExt: ", result, "\n");
return result;
}

```

HCICatcCreateConnectionExt

HCICatcCreateConnectionExt (AddressPacketTypePageScanRepetitionModePageScanModeClockOffsetAllowRoleSwitchNumberOfPackets, TimeLength, DataErrorType, ErrorMask, TxJitter)

Comments

This command supports TCI functionality that requires BtTrainer to be connected as a master device to the IUT. This command will initiate a base-band connection process and perform the test defined by the parameters.

Parameter	Meaning	Value	Comments
Device Address			
PacketType			Values defined for HCI_CreateConnection command
PageScanRepetitionMode			Values defined for HCI_CreateConnection command
PageScanMode			Values defined for HCI_CreateConnection command
ClockOffset			Values defined for HCI_CreateConnection command

User's Manual

AllowRoleSwitch			Values defined for HCI_CreateConnection command. Value will be 0x00 (not allow role switch) if ErrorType is not 0.
NumberOfPackets		0x0001 -- 0xFFFF	Number of packets to transmit. On completion, only NULL-POLL packets will be transmitted to maintain the connection
TimeLength		0x0000 -- 0xFFFF (ms)	Amount of time to send data packets. After <i>TimeLength</i> , only NULL-POLL packets will be transmitted to maintain the connection. <i>TimeLength</i> will override <i>NumOfPackets</i> parameter if <i>TimeLength</i> value is not zero.
Data			Cannot be longer than the maximum packet size of the selected $\min(\text{PacketType}, \text{maxHCICommandLength})$.
ErrorType			Possible values listed in the table below
ErrorMask		0 - Random error distribution >0 - Set bits indicate error positions	If <i>ErrorType</i> is not zero, this parameter is valid. It defines error distribution.
TxJitter		0 - No jitter is added	Random jitter between --10 and +10us is added to the nominal 1250 us Master Tx timing. The value is 2's compliment Example of Jitter values are: 0-10 + Jitter 0 to10, w/o 0.5us 64-74+ Jitter 0 to10, w/ 0.5us -64 to -74- Jitter -1 to -10, w/ 0.5us- -1 to -10 - Jitter -1 to -10, w/o 0.5us
L_CH	1	L_CH=2b'01	Defines payload header L_CH value and type of the payload to be sent.
		Continuation fragment of an L2CAP message (Default value)	
	2	L_CH=2b'10	Start of an L2CAP message or no fragmentation

3 L_CH = 11 - LMP
Message

ErrorType

0x0000	No error
0x0001	FEC 1/3 -- uncorrectable packet header
0x0002	FEC 2/3 -- uncorrectable payload. Special case is for DV payload
0x0004	Explicit NAK
0x0008	SEQN. See TP/PROT/ARQ/BV-18-C test case
0x0010	FLOW control
0x0020	Same SEQN value. See TP/PROT/ARQ/BV-23-C test case
0x0040	Wrong UAP
0x0080	Wrong AM_ADDR of the slave
0x0100	Deadlock avoidance according to the LMP test case TP/LIH/BV-81-C

Return value

- "Success"
- "Failure"
- "Invalid Parameter"
- "Already connected"
- "Timed out"

Example

```
#
# HCI_CatcCreateConnectionExt
#

Main()
{
    Device                = '008037163567';
    PacketType            = ["DM3"];
    PageScanRepetitionMode = 0x0;
    PageScanMode          = 0x0;
    ClockOffset           = 0;
    AllowRoleSwitch       = 1;
    NumberOfPackets       = 1000;
    TimeLength            = 0;
    Data                  = 0x010203;
    ErrorType              = 0x0000;
    ErrorMask              = 0x0000;
```

```

TxJitter                = 0x01;
L_CH                    = 2;

result = HCICatcCreateConnectionExt (Device,
                                     PacketType,
                                     PageScanRepetitionMode,
                                     PageScanMode,
                                     ClockOffset,
                                     AllowRoleSwitch,
                                     NumberOfPackets,
                                     TimeLength,
                                     Data,
                                     ErrorType,
                                     ErrorMask,
                                     TxJitter,
                                     L_CH );

Trace("HCICatcCreateConnectionExt: ", result, "\n");

result = Disconnect (Devices [0]);
return result;
}

```

HCICatcSetClock

HCICatcSetClock (LocalClock)

Parameter	Meaning	Default Value	Comments
Device			

Return value

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not Connected"

Comments

This command causes sending LMP_Test_Activate PDU from the Tester to the DUT.

Example

```

Devide = '010203040506';
result = HCICatcEnterTestMode (Device);
Trace (result);

```

HCICatcSetBdAddr

HCICatcSetBdAddr (Address)

Parameter	Meaning	Default Value	Comments
Address			

Return value

- "Success"
- "Failure"

Comments

The command will set the value of the local Bluetooth device address.

Example

```
Address = '010203040506';
result = HCICatcSetBdAddr (Address);
Trace (Address);
```

HCICatcEnterTestMode

HCICatcEnterTestMode ()

Parameter	Meaning	Default Value	Comments
Device			

Return value

- "Success"
- "Failure"
- "Failed: Device not found"
- "Not Connected"

Comments

This command causes sending LMP_Test_Activate PDU from the Tester to the DUT.

Example

```
Devide = '010203040506';
result = HCICatcEnterTestMode (Device);
Trace (result);
```

HCICatcInquiry

HCICatcInquiry (TestScenario, NumberOfTimes)

Parameter	Meaning	Default Value	Comments
TestScenario			
NumberOfTimes			

TestScenario

0x00	Exit the Inquiry test mode
0x01	Perform Inquiry procedure according to TP/PHYS/INQ/BV-10-C test cases

Return value

- "Success"
- "Failure"
- "Invalid Parameter"
- "Timed out"

Comments

This command supports TCI functionality that requires BtTrainer to initiate a modified Inquiry procedure to the IUT, as required by the Bluetooth Baseband test cases

Used in Test Cases

Test Case	Description
TP/PHYS/INQ/BV-10-C	Inquiry response/79 channels

Example

```
result = HCICatcInquiry (TestScenario, NumberOfTimes);
Trace (result);
```

HCICatcInquiryScan

HCICatcInquiryScan (TestScenario, Ninquiry)

Parameter	Meaning	Default Value	Comments
TestScenario	0x00 -Exit Inquiry Scan test mode 0x01- Disable responding to incoming inquiry ID packets. Perform according to TP/PHYS/INQ/BV-01-C or TP/PHYS/INQ/BV-02-C according to the current Country Code settings. 0x02-0xFF- Reserved		
Ninquiry	Number of train repetitions. Default is 256		

Return value

- "Success"
- "Failure"
- "Invalid Parameter"
- "Timed out"

Comments

This command supports TCI functionality that requires BtTrainer to enter modified Inquiry Scan mode, and respond to the IUT Inquiry attempts as required by the Bluetooth Baseband test cases.

This command turns off the scan enable mode. You should call Write_Scan_Enable to restore the value.

Used in Test Cases

Test Case	Description
TP/PHYS/INQ/BV-01-C	Inquiry hop seq/79 channel. Disable responding to incoming inquiry ID packets as defined in this test case

Example

```
result = HCICatcInquiryScan (TestScenario,
Ninquiry); Trace (result);
```


HCICatcPage

HCICatcPage (Address, TestScenario, NumberOfTimes)

Parameter	Values	Description
Device Address		Bluetooth address of the paged device.
TestScenario	0x00	Return to use the normal Page
	0x01	The tester pages the IUT only in the first half of the Tx time slot. After receiving response, the tester does not send an FHS packet. The test continues according to TP/PHYS/PAG/BV-10-C or TP/PHYS/PAG/BV-11-C test case according to the current Country Code.
	0x02	The tester pages the IUT only in the second half of the Tx time slot. After receiving response, the tester does not send an FHS packet. The test continues according to TP/PHYS/PAG/BV-12-C or TP/PHYS/PAG/BV-13-C test case according to the current Country Code.
	0x03	The tester pages the IUT until the response ID is received. The tester does not respond with FHS, but waits 16 slots, then 8 slots, and then an additional randomly chosen 0-1023 slots. Perform the test according to TP/PHYS/PAG/BV-14-C test case and current page scheme.
	0x04	The tester pages the IUT until the response ID is received. The tester does not respond with FHS, but waits 16 slots, then 8 slots, and then an additional randomly chosen 0-2047 slots. Perform the test according to TP/PHYS/PAG/BV-16-C test case and current page scheme.
	0x05	The tester pages the IUT until the response ID is received. The tester does not respond with FHS, but waits 16 slots, then 8 slots, and then an additional randomly chosen 0-4095 slots. Perform the test according to TP/PHYS/PAG/BV-18-C test case and current page scheme.
	0x06-0xFF	Reserved
NumberOfTimes		Number of times to repeat the test.

Return value

- "Success"
- "Failure"
- "Invalid Parameter"
- "Timed out"

Comments

This command supports TCI functionality that requires BtTrainer to initiate a modified Page procedure to the IUT, as required by the Bluetooth Base-band test cases.

User's Manual

Used in Test Cases

Test Case	Description
TP/PHYS/PAG/BV-10-C	Page response 1/1 slot/79 channel.
TP/PHYS/PAG/BV-12-C	Page response 1/2 slot/79 channel
TP/PHYS/PAG/BV-14-C	Page scan interval R0.
TP/PHYS/PAG/BV-16-C	Page scan interval R1.
TP/PHYS/PAG/BV-18-C	Page scan interval R2.

Example

- Address = '010203040506';
- TestScenario = 1;
- NumberOfTimes = 100;
- result = HCICatcPage(Address, TestScenario, NumberOfTimes);
- Trace(result);

HCICatcPageScan

HCICatcPageScan (TestScenario, nPage)

Parameter	Meaning	Default Value	Comments
TestScenario	See table below		
Npage	Specifies Npage of the paging device		

Test Scenario values

0x00	Return to use the normal Page Scan
0x01	Disable responding to incoming page ID packets according to the TP/PHYS/PAG/BV-01-C test cases
0x02	Respond to the first ID packet and the second received FHS packet from the Paging device according to TP/PHYS/PAG/BV-03-C test
0x03	Respond to the second ID packet and the second received FHS packet from the Paging device according to TP/PHYS/PAG/BV-05-C test cases
0x04-0xFF	Reserved

Return value

- "Success"
- "Failure"
- "Invalid Parameter"

Comments

This command supports TCI functionality that requires BtTrainer to enter modified Page Scan mode, and respond to the IUT Page attempts as required by the Bluetooth Baseband test cases

Used in Test Cases

Test Case	Description
TP/PHYS/PAG/BV-01-C	Page hop seq/79 channel. Tester does not respond to the Page ID packets as defined in this test case.
TP/PHYS/PAG/BV-03-C	Page response to 1st msg/79 channel. Tester responds to the first Page ID and does not respond to the first FHS packet as defined in this test case.
TP/PHYS/PAG/BV-05-C	Page response to 2nd msg/79 channel. Tester does not respond to the first Page ID and the first FHS packet as defined in this test case.

Example

```
TestScenario = 1;
Npage = 256;
result = HCICatcPageScan(TestScenario,Npage);
Trace(result);
```

HCICatcTestControlMaster

```
HCICatcTestControlMaster((Address, TestScenario,
HoppingMode, TxFrequency, RxFrequency, PowerControlMode,
PollPeriod, PacketType, NumberOfPackets, TimeLength,
Data, ErrorType, ErrorMask, TxJitter)
```

Comment

This command supports TCI functionality that requires BtTrainer to be in the Test Mode as a master device. It will generate LMP_Test_Control to enter the Test Mode, if necessary, and will start performing the test, as specified by the parameters.

Parameter	Values	Description
Device 2 Bytes	Example: 0x0001	HCI ACL Connection handle

User's Manual

Test Scenario 1 Byte	0x00	Pause Test Mode	Values defined for LMP_TestControl command. This parameter defines the transmitted data type and the loopback mode.
	0x01	Transmitter test – 0 pattern	
	0x02	Transmitter test – 1 pattern	
	0x03	Transmitter test – 1010 pattern	
	0x04	Pseudorandom bit sequence	
	0x05	Closed Loop Back – ACL packets	
	0x06	Closed Loop Back – SCO packets	
	0x07	ACL Packets without whitening	
	0x08	SCO Packets without whitening	
	0x09	Transmitter test – 1111 0000 pattern	
	0xFF	Exit Test Mode	
	0x0A-0xFF	Reserved	
Hopping Mode 1 Byte	0x00	Tx/Rx on a single frequency	Values defined for LMP_TestControl command
	0x01	Europe/USA	
	0x05	Reduced hopping	
	0x06-0xFF	Reserved	
TxFrequency 1 Byte	Example: 0x01		Values defined for LMP_TestControl command. Valid only in a single frequency mode.
RxFrequency 1 Byte	Example: 0x02		Values defined for LMP_TestControl command. Valid only in a single frequency-hopping mode.

User's Manual

PowerControl Mode 1 Byte	0	Fixed Tx output power	Values defined for LMP_TestControl command
	1	Adaptive power control	
PollPeriod** 1 Byte	Example: 1 - POLL is sent every master-to-slave slot		Values defined for LMP_TestControl command. Units: 1.25 ms In Closed Loopback mode, set to 0 (not applicable)
PacketType 1 Byte	POLL	0x01	Values defined for LMP_TestControl command
	NULL	0x00	
	DM1	0x03	
	HV1	0x05	
	HV2	0x06	
	HV3	0x07	
	DV	0x08	
	DH1	0x04	
	DM3	0x0A	
	DH3	0x0B	
	DM5	0x0E	
	DH5	0x0F	
	AUX1	0x09	
		Reserved	
NumOf Packets 2 Bytes	0x0001 – 0xFFFF		Number of data packets to transmit. After <i>NumOfPackets</i> are transmitted, only NULL or POLL packets will be sent to maintain the connection.
Time Length 2 Byte	0x0000 – 0xFFFF (ms)		Amount of time to send data packets. After <i>TimeLength</i> , only NULL-POLL packets will be transmitted to maintain the connection. <i>TimeLength</i> will override <i>NumOfPackets</i> parameter if <i>TimeLength</i> value is not zero.

User's Manual

PacketLength 2 Bytes	0x0000 - <i>MaxPacketSize</i>	Cannot be longer than the maximum packet size of the selected min(<i>PacketType</i> , <i>maxHCICCommandLength</i>). Currently it's set to 224 Bytes. In Closed Loopback mode, set to 0 (not applicable)
Data From 0 to MaxPacket Size Bytes		If <i>TestScenario</i> does not require a predefined data pattern, this parameter will be used. Otherwise, it'll be ignored. Data length is defined by the <i>PacketLength</i> parameter. If it is 0, this parameter will be ignored.
ErrorType 2 Bytes		
	0x0008	HEC error injection
	0x0010	CRC error injection
	0x0040	FEC 2/3 error injection
	0x0080	FEC 1/3 header error injection
	0x0100	FEC 1/3 payload error injection
	0x0200	Wrong AM_ADDR of the slave.
	0x0000	No error
	0x0201 – 0xFFFF	Reserved

User's Manual

ErrorMask 2 Bytes	0	Random error distribution	<p>If <i>ErrorType</i> is not zero, this parameter is valid. It defines error distribution.</p> <p>For FEC 2/3, random error distribution, one error bit will be inserted in every 15-bit packet. Valid for DM packet types</p> <p>For FEC 1/3 header, random error distribution, one error bit will be inserted in every 3-bit packet. Valid for all packet types with the header</p> <p>For FEC 1/3 payload, random error distribution, one error bit will be inserted in every 3-bit packet. Valid for DH1 packet type.</p>
	>0	Set bits indicate error positions within 15-bit mask for FEC 2/3	
Tx Jitter 1 Byte	0	No jitter is added	<p>Random jitter between -10 and +10us is added to the nominal 1250 us Master Tx timing. The value is 2's compliment. 0.5us is supported. For example, 64 will give a jitter of 0.5us, 70 will give jitter of 5.5us</p> <p>Example of Jitter values are:</p> <p>0-10 Jitter 0 to10, w/o 0.5us</p> <p>64-74 Jitter 0 to10, w/ 0.5us</p> <p>-64 to -74 Jitter -1 to -10, w/ 0.5us</p> <p>-1 to -10 Jitter -1 to -10, w/o 0.5us</p>
	N= 0x00 0xFF	Random jitter up to N us is added.	
	Other	Reserved	

**Comply with the Loopback Mode restrictions in I:4, section 2.2 of the 1.1 Bluetooth spec.

Return

- "Success"
- "Failure"
- "Device not connected"
- "Invalid parameters"

Return value

² "Success"

² "Failure"

User's Manual

- ² "Invalid Parameter"
- ² "Not connected"
- ² "Failed: Device not found"

Used in Test Cases

Test Case	Description
TP/PHYS/FRE/BV-01 -C	79 channel hopping sequence. Tester transmits POLL packets every master-to-slave slot. Clock is set to include wrap-around
TP/PHYS/FRE/BV-03 -C	23 channel hopping sequence. Tester transmits POLL packets every master-to-slave slot. Clock is set to include wrap-around
TP/PHYS/TRX/BV-02 -C	Slave Tx timing. Random 0-10 us jitter is added to the Master Tx timing
TP/PHYS/TRX/BV-02 -E	Slave Tx timing. Random 0-10 us jitter is added to the Master Tx timing
TP/PROT/COD/BV-1 1-C	Erroneous slave address. AM_ADDR is corrupted.
TP/PROT/COD/BV-1 2-C	Correctable packet header – FEC13
TP/PROT/COD/BV-1 4-C	Correctable error HV1 payload – FEC13
TP/PROT/COD/BV-1 6-C	Correctable error DM1 payload – FEC23

Example

```

Device      = '008037163567';
TestScenario = 0x07;
HoppingMode = 1;
TxFrequency = 0;
RxFrequency = 0;
PowerControlMode = 0;
PollPeriod  = 0x40;
PacketType  = ["DH1"];
NumOfPackets = 100;
TimeLength  = 0;
Data        = 0x010203;
ErrorType   = 0x0000;
ErrorMask   = 0;
TxJitter    = 0xFE;

result = HCICatcTestControlMaster(
    Device,
    TestScenario,
    HoppingMode,
    TxFrequency,

```

```

        RxFrequency,
        PowerControlMode,
        PollPeriod,
        PacketType,
        NumOfPackets,
        TimeLength,
        Data,
        ErrorType,
        ErrorMask,
        TxJitter );

    Trace("HCICatcTestControlMaster: ", result, "\n");
    return result;

```

HCICatcEnterTestMode

HCICatcEnterTestMode (Address)

Parameter	Meaning	Default Value	Comments
Address	Bluetooth address of remote device		

Return value

- "Success"
- "Failure"
- "Note found"
- "Not connected"

Comments

This command will start a modified ACL disconnection procedure in order to verify that the IUT closes the link according to the TP/LIH/BV-04-C test case. During the procedure, LMP_Detach is sent, and after 3 seconds LMP_Features_Req PDUs is sent.

Example

B.12 BTTracer Commands

BTTracerStartRecording

BTTracerStartRecording()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- “Success”
- “Failure”

Comments

This command is used to start a BTTracer recording.

Example

```
BTTracerStartRecording();
```

BTTracerStopRecording

BTTracerStopRecording()

Parameter	Meaning	Default Value	Comments
N/A			

Return value

- “Success”

Comments

This command is used to stop a BTTracer recording.

Example

```
BTTracerStopRecording();
```


Appendix C: 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 BTTrainer 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.

C.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.

C.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:

User's Manual

Hexadecimal numbers must be preceded by 0x: 0x2A, 0x54, 0xFFFFFFFF01

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 <small>**Note that this does not represent the integer 256, but only the characters that make up the number.</small>
"abcd!\$%&*"	abcd!\$%&*
"June 26, 2001"	June 26, 2001
"[1, 2, 3]"	[1, 2, 3]

Escape Sequences

These are the available escape sequences in CSL:

<u>Character</u>	<u>Escape Sequence</u>	<u>Example</u>	<u>Output</u>
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\n to 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;
```

C.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.

C.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.

C.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 C.6, “Operators,” on page 297. 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 $x \geq 5$. 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.

C.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 300. 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

User's Manual

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.
-	Subtraction	Integer-integer	Integer	3 - 1 = 2
		String-list	List	"one" + ["two"] = ["one", "two"]
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

User's Manual

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)

User's Manual

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.
+=	Addition assignment	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)

User's Manual

Operator Symbol	Description	Operand Types	Result Types	Examples
Assignment Operators (continued)				
<code>&=</code>	Bitwise AND assignment	Integer-integer	Integer	<code>a = 0b11111110</code> <code>a &= 0b01010101 = 0b01010100</code>
<code>^=</code>	Bitwise exclusive OR assignment	Integer-integer	Integer	<code>e = 0b11111110</code> <code>e ^= 0b01010101 = 0b10101011</code>
<code> =</code>	Bitwise inclusive OR assignment	Integer-integer	Integer	<code>i = 0b11111110</code> <code>i = 0b01010101 = 0b11111111</code>
List Operators				
<code>sizeof()</code>	Number of elements	Any	Integer	<code>sizeof([1, 2, 3]) = 3</code> <code>sizeof('0011223344') = 5</code> <code>sizeof("string") = 6</code> <code>sizeof(12) = 1</code> <code>sizeof([1, [2, 3]]) = 2</code> *Note: the last example demonstrates that the <code>sizeof()</code> operator returns the shallow count of a complex list.
<code>head()</code>	Head	List	Any	<code>head([1, 2, 3]) = 1</code> *Note: the Head of a list is the first item in the list.
<code>tail()</code>	Tail	List	List	<code>tail([1, 2, 3]) = [2, 3]</code> *Note: the Tail of a list includes everything except the Head.

Operators (Continued)

C.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");  
}
```

C.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

C.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

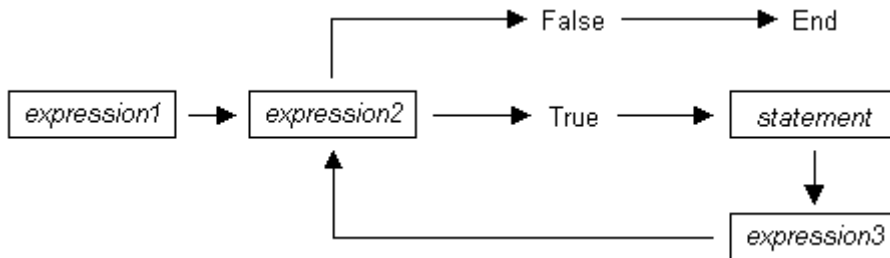
2, 3, 4,

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.

C.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"`.

C.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.

C.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 <i>string</i>			
arg_list <i>list</i>			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 <i>string</i>			
value <i>string</i> or <i>integer</i>			

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:

<u>Code</u>	<u>Type</u>	<u>Output</u>
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.

User's Manual

- 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 <i>list</i> , <i>raw</i> , or <i>integer</i>			Can be an integer value (4 bytes) or a list of integers that are interpreted as bytes.
bit_offset <i>integer</i>	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
<code>arg any</code>			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]
```

