Qualcomm Technologies International, Ltd.

Confidential and Proprietary – Qualcomm Technologies International, Ltd.

(formerly known as Cambridge Silicon Radio Ltd.)

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to: DocCtrlAgent@qualcomm.com.

Restricted Distribution: Not to be distributed to anyone who is not an employee of either Qualcomm Technologies International, Ltd.or its affiliated companies without the express approval of Qualcomm Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies International, Ltd.

Any software provided with this notice is governed by the Qualcomm Technologies International, Ltd. Terms of Supply or the applicable license agreement at https://www.csrsupport.com/CSRTermsandConditions.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. All Qualcomm Incorporated trademarks are used with permission. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer (“export”) laws. Diversion contrary to U.S. and international law is strictly prohibited.

Qualcomm Technologies International, Ltd.
Churchill House
Cambridge Business Park
Cambridge, CB4 0WZ
United Kingdom

CSR

Push every boundary.™

BlueCore®

BTCli

User Guide

Issue 3

Document History

Revision	Date	History
1	30 AUG 07	Original publication of this document
2	26 SEP 14	Updated to new CSR style
3	29 APR 15	Updated for BlueSuite 2.6.0

Contacts

General information	www.csr.com
Information on this product	sales@csr.com
Customer support for this product	www.csrsupport.com
More detail on compliance and standards	product.compliance@csr.com
Help with this document	comments@csr.com

Trademarks, Patents and Licences

Unless otherwise stated, words and logos marked with ™ or ® are trademarks registered or owned by CSR plc and/or its affiliates.

Bluetooth® and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR.

Other products, services and names used in this document may have been trademarked by their respective owners.

The publication of this information does not imply that any licence is granted under any patent or other rights owned by CSR plc or its affiliates.

CSR reserves the right to make technical changes to its products as part of its development programme.

While every care has been taken to ensure the accuracy of the contents of this document, CSR cannot accept responsibility for any errors.

Life Support Policy and Use in Safety-critical Compliance

CSR's products are not authorised for use in life-support or safety-critical applications. Use in such applications is done at the sole discretion of the customer. CSR will not warrant the use of its devices in such applications.

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.

Contents

Tables, Figures and Equations

1. Introduction

This document describes the **BTCli** tool, which is part of the standard installation of CSR's **BlueSuite** PC tools. **BTCli** is a tool which is useful as an interface to CSR's BlueCore ICs at the HCI level. **BTCli** cannot emulate Bluetooth profiles at a level above HCI.

BTCli is tool designed to send specific commands to BlueCore in an effort to test specific scenarios. Application level testing is best left for other tools, such as the Bluetooth SIG's Profile Testing Suite (PTS).

This document does not describe all possible commands that the user can access with **BTCli**, but provides guidance on how to discover those commands and describes how to perform common tests using **BTCli**.

2. Overview of BTCli Capabilities

2.1. Host Interface Options

BTCli supports the following UART interfaces:

- Raw UART (H4)
- The HCI 3-Wire interface (H5)
- CSR's BlueCore Serial Protocol (BCSP)
- CSR's raw UART with Deep Sleep (H4DS)

BTCli also supports the USB host interface and the SDIO interface.

If during start up or at some point during the test the host interface fails, an error message is displayed indicating the host transport has failed. **BTCli** makes no effort to reconnect or to restart the host transport. The user can type **restart** to restart the interface, or if necessary close the program, reset the device, and try again.

2.2. Command Types

BTCli is capable of sending and processing:

- HCI commands, events, and data as defined in the Bluetooth core specifications
- BCCMD commands as defined by CSR's BCCMD protocol and command set documentation. See *HQ and BCCMD Commands*.
- **BTCli** supports Persistent Store (PS) commands (a subset of the BCCMD commands) in which most PS Keys may be accessed via a friendly text name and do not require the specific key's PS Key number.
- Low level radio test commands, similar to those found in **BlueTest3** (another tool found in the **BlueSuite** tools installation) are also available. Radio test command support is provided and is another friendly-name subset of the Radio Test BCCMD commands. See *HQ and BCCMD Commands*.
- I^{2}C test commands and EEPROM commands, which are a subset of the BCCMD command set, are also supported with a friendly-name interface.
- FM radio support is provided for relevant BlueCore devices.
- Raw byte sequences are supported in the event that the specific version of **BTCli** being used does not support a new or custom command.
- Virtual Machine (VM) commands are supported in the event that command testing is required for an on-chip VM application.
- HQ events are supported, which is the CSR-specific error reporting mechanism for BlueCore.

2.3. Session and Data Logging

It is possible to have **BTCli** create a session log of all commands, events, and data sent/received during the test session.

It is possible to capture SCO data into its own file and each SCO stream (if multiple SCO connections are created) can be routed to their own separate log files. Table 3.1 describes the specifics of data logging using command line parameters

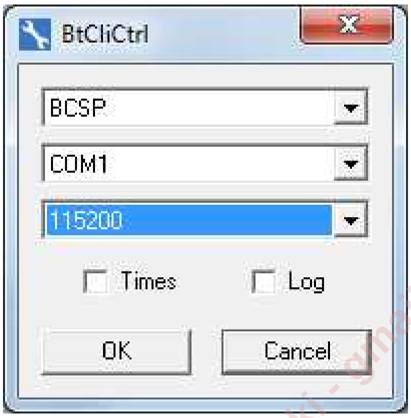
3. Using BTCli

3.1. Launching BTCli

BTCli can be launched from the PC's **Start** menu by navigating to the CSR **BlueSuite** menu and selecting **BTCli**.

3.2. Selecting the Host Transport

Figure 3.1 shows the **BtCliCtrl** window used to select the correct host interface. The window also provides timestamp and log options.



BtCliCtrl

BCSP

COM1

115200

□ Times □ Log

Figure 3.1: BtCliCtrl Window

The **BtCliCtrl** window allows the user to select from the available host transports supported by **BTCli**, i.e:

- BCSP
- H4
- H4DS
- USB
- H5
- SDIO

The host transport is configured using PS Keys. The procedure to change the host transport is outlined in greater detail in the *Casira User Guide*.

If H4, H4DS, H5, or BCSP is selected, the second option menu allows the user to select a COM port to use for communication. The third option menu allows the user to select a baud rate up to 1382400 baud. If the test requires a faster baud rate or a baud rate not specified in the drop menu, then the command line option must be used to launch **BTCli**. Command line options are described in section3.3.

If USB is selected, the second option menu is automatically populated with all the USB devices attached to the computer that are currently using CSR's USB driver. If a BlueCore has been connected to the system via USB and is not in this menu, then check the device is configured for USB and that the CSR USB driver is being used.

If the **Times** option is selected, timestamps are added to command and event displays.

If **Log** is selected, then the output of **BTCli** is placed into a log file. The default name of the log file is `btcli.log`.

Note:

> If a different log file name is required, then the best way to accomplish this is to launch **BTCli** from the command line. Command line operation is described in section 3.3 and the command line option is described specifically in the description of the `-l` parameter in Table 3.1.

3.3. Command Line Options

The official usage statement for the **BTCli** command line displays two methods for connection.

For BlueCore devices, the following applies:

```
        btcli [<transport>] [<port>] [<bps>] [-pe|-po|-pn] [-B0|-B1|-B2]
              [-1|-2|-3|-4|-5|-6|-7] [-a] [-A] [-c] [-C] [-DP] [-e] [-fON|-fOFF]
              [-F<path>] [-g] [-HW<waketime>] [-l[<file>]] [-rl<n>] [-rt<timeout>]
              [-s] [-s<n>] [-s[<n>]<file>] [-S<file>] [-t] [-T] [-w] [-x<file>] [-X]
              [-accmdserver[<port>]] [-help]
```

For HydraCore devices, the following applies:

```
        btcli ptap [<curatorHifId> [<secondaryHifPortName>]]
```

The listed BlueCore parameters (e.g. `-x<file>`), other than `<transport>`, `<port>` and `<bps>`, may also be used in conjunction with the HydraCore parameters when connecting to a HydraCore device. (Any such options not relevant to the host transport in use will be ignored, as is also the case for BlueCore devices.)

Table 3.1 defines the various parameters of the **BTCli** command line. Note that in the case of flags taking arguments, there is no space between the flag and the argument, e.g. `-lLogFile.txt`.

Parameter	Definition
`[<transport>]`	The transport parameter defines which host transport **BTCli** should use to connect with BlueCore. Available options are: `bcsp`, `h4`, `h4ds`, `h5`, `usb` BCSP is used as the default transport if no option is given for this parameter.
`[<port>]`	The port parameter defines the communication port **BTCli** should use to connect with BlueCore. The choice of the port depends on the host transport used (UART-based or USB). If a UART-based host transport is used (i.e.BCSP, H4, H4DS, or H5), then **BTCli** uses a COM port such as COM1. If no COM port is specified in the command line, then **BTCli** attempts COM1 through COM20 in sequence before giving up. If the USB host transport is used, **BTCli** attempts to open `\\.\csr0` as the default USB port. If the device is connected to another USB port, that port must be explicitly specified.
`[<bps>]`	This parameter defines the baud rate for the UART-based host transport defined in the `<transport>` parameter. It is possible to specify any baud rate, but if no value is specified the default is 38400 bps. (For a successful connection, a baud rate supported by the relevant device must be used.) This parameter is not required for the USB host transport.
`[<curatorHifId>]`	For a HydraCore device, specifies the hif id of the target curator for ptap e.g. 0 (default).
`[<secondaryHifPortName>]`	For a HydraCore device, specifies the secondary hif port name for ptap e.g. COM1 (default).
`[-pe|-po|-pn]`	Specifies the parity for a UART-based transport (even, by default).
`[-B0|-B1|-B2]`	Specifies the stop bits for a UART-based transport.
`[-1|-2|-3|-4|-5|-6|-7]`	Specifies the window size (BCSP only).

Parameter	Definition
[-a]	The [-a] parameter behaves the same way as the [-s] parameter in that ACL data can be prevented from scrolling on the screen. This parameter does not affect file transfers, including display of bad packets received on connection handles used for file transfers. File transfers are described in section 4.4 of this document.
[-A]	The [-A] parameter causes **BTCli** to display any incoming byte streams in ASCII format.
[-c]	The [-c] parameter is only useful for the BCSP host transport and specifies not to use Cyclic Redundancy Checks (CRC). BCSP use of CRC is described in BlueCore Serial Protocol (BCSP).
[-C]	Disable CSR extensions
[-DP]	Enable auto protocol detect (H4 only)
[-e]	Disables the display of variable substitution expansion. See section 4.1 for information on variables.
[-fON|-fOFF]	Specifies whether to use hardware flow control for a UART-based transport.
[-F<path>]	Specifies the path for FSM data.
[-g]	Display GPS data.
[-HW<waketime>]	Specify the wakeup time (h4ds only)
[-l<file>]	This parameter specifies a session log file should be generated. If no filename is specified (i.e. -l on its own), then a default filename btcli.log (within the subfolder "CSR Ltd\btcli" of the user's local application data folder, e.g. "C:\Users\<username>\AppData\Local\CSR Ltd\btcli\btcli.log") is used. If a filename is supplied with no path, e.g. -lLogfile.txt, then the session logfile will be created within the current working directory. **Note:** Any existing file is appended to rather than overwritten. **BTCli** should be exited cleanly (i.e. by typing quit) to ensure the session log file is written correctly.
[-rl<n>]	The [-rl<n>] parameter only applies when BCSP is selected as the host transport and specifies the BCSP retry limit size. BCSP retry limit is described in BlueCore Serial Protocol (BCSP).
[-rt<timeout>]	The [-rt<timeout>] parameter is only useful for the BCSP host transport and specifies the BCSP resend timeout. BCSP resend timeout is described in BlueCore Serial Protocol (BCSP).
[-s]	The [-s] parameter is simply a variation of the [-s[<n>]<file>] parameter. It configures **BTCli** to not display SCO data on the screen even if no filename is specified. **Note:** Bad packets are always displayed regardless of this parameter's value.

Parameter	Definition
`[-s<n>]`	Deprecated.
`[-s[<n>]<file>]`	If a SCO connection is established in **BTCli**, by default the SCO data packets are displayed in the **BTCli** window. The preferred method is to route this data to a file for post processing. The `[-s[<n>]<file>]` parameter allows the incoming SCO data to be routed to a file rather than displayed on the screen, for SCO channel `<n>`, where `<n>` may take value 0, 1 or 2. It's implicit from the format of this parameter that a single digit of value 0, 1 or 2, immediately following `-s`, will be interpreted as a SCO channel identifier rather than as the beginning of a file name. If no SCO channel is specified (i.e. no `<n>` is specified), then all SCO data on SCO channel 0 is routed to the file.
`[-S<file>]`	Loads symbol data from `<file>`.
`[-t]`	The `[-t]` parameter enables timestamps.
`[-T]`	Tunnel BlueCore channels (BCSP only)
`[-w]`	The `[-w]` parameter causes **BTCli** to automatically enable all link policy settings on an ACL link after it is established. Link policies are the same as those defined in the HCI section of the Bluetooth Core Specifications and include low power modes, role switching, etc.
`[-x<file>]`	Execute script `<file>`. See section 4.2 for details of scripting.
`[-X]`	Quiet mode
`[-accmdserver[<port>]]`	Specify the ACCMD server port

Table 3.1: BTCli Command Line Options

3.4. BTCli Command Support

BTCli provides support for:

- HCI commands and events
- BCCMD
- Persistent Store
- HQ events
- Radio Test commands
- Virtual Machine messages.

Sections to 3.4.1 to 3.4.9 define this support in greater detail.

3.4.1. HCI Command Support

3.4.1.1. Command Naming Conventions

In nearly every case, an HCI command can be given using the first letter of each word in the command's name (as specified in the HCI section of the Bluetooth Core Specifications). For instance, the HCI `read_local_version_information` command is `rlvi`.

The exceptions are:

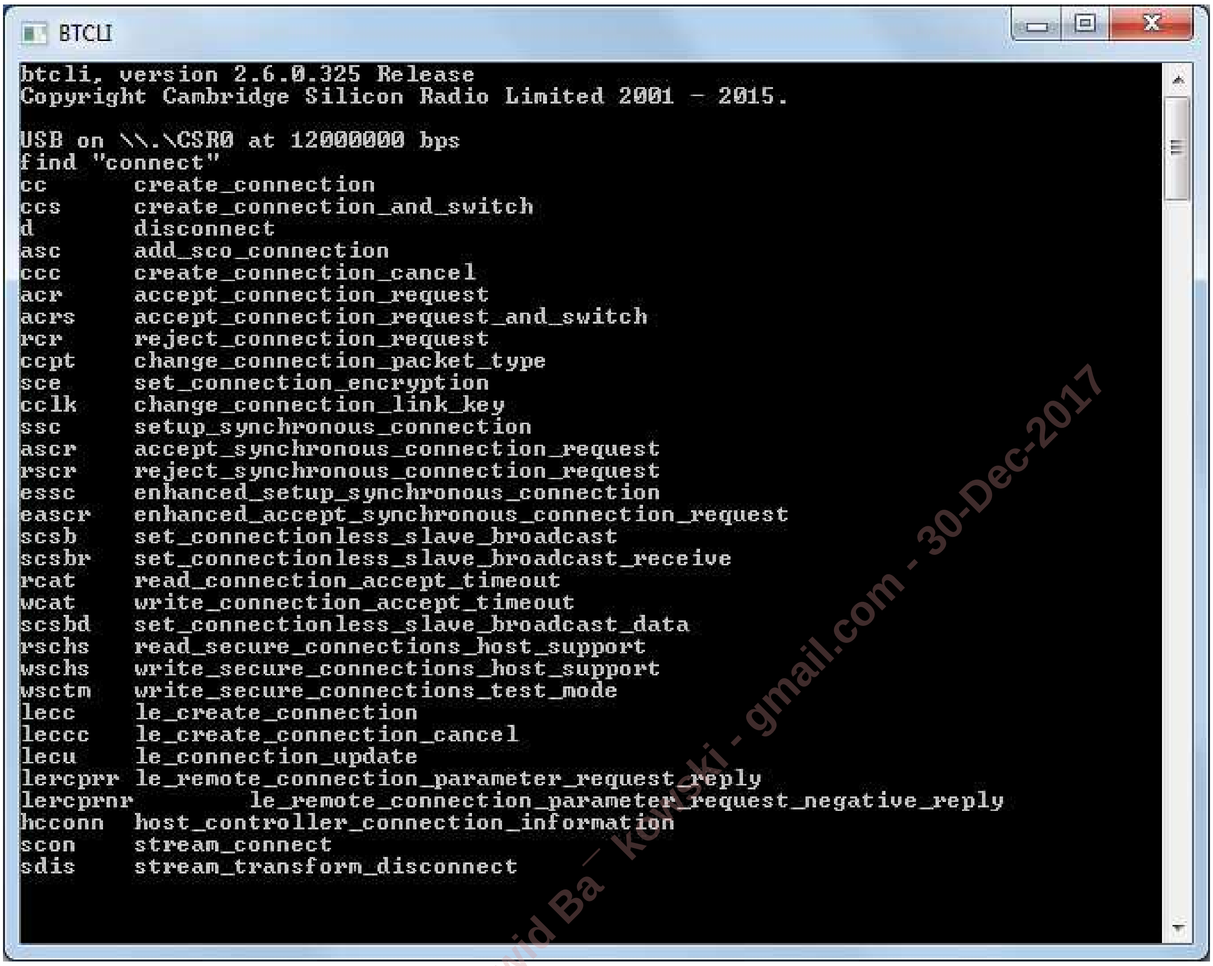
- `rpint` and `wpint` for Read/Write `PIN_Type`
- `rstfcc` for `reset_failed_contact_counter`
- `rba` for Read `bluetooth_device_address`
- When the command name contains an 'and' it is ignored e.g. `create_connection_and_switch` is `ccs` not `ccas`.
- When the command name begins with 'io' the 'io' remains as 'io' e.g. `io_capability_response` is `iocr`, not `icr`).
- When the command name begins with 'le' the 'le' remains as 'le' e.g `le_set_scan_enable` is `lesse` not `lsse`.
- The `nop` command does not have an abbreviation

3.4.1.2. Tips for Finding Specific HCI Commands

BTCli includes a command called find to help find specific commands. Typing `find` on the command line lists all commands including non-HCI commands such as BCCMD and PS Key commands. This list is lengthy and a specific command may still be hard to find.

However, `find` can be used in association with a string to narrow the search. For instance typing `find "connect"` (including the quotes) displays all commands that contain the substring "connect".

Figure 3.2 shows an example output, for the find `"connect"` command.



```
btcli, version 2.6.0.325 Release
Copyright Cambridge Silicon Radio Limited 2001 - 2015.

USB on \\.\CSR0 at 12000000 bps
find "connect"
cc      create_connection
ccs     create_connection_and_switch
d       disconnect
asc     add_sco_connection
ccc     create_connection_cancel
acr     accept_connection_request
acrs    accept_connection_request_and_switch
rcr     reject_connection_request
ccpt    change_connection_packet_type
sce     set_connection_encryption
cclk    change_connection_link_key
ssc     setup_synchronous_connection
ascr    accept_synchronous_connection_request
rscr    reject_synchronous_connection_request
essc    enhanced_setup_synchronous_connection
eascr   enhanced_accept_synchronous_connection_request
scsb    set_connectionless_slave_broadcast
scsbr   set_connectionless_slave_broadcast_receive
rcat    read_connection_accept_timeout
wcat    write_connection_accept_timeout
scsbd   set_connectionless_slave_broadcast_data
rschs   read_secure_connections_host_support
wschs   write_secure_connections_host_support
wsctm   write_secure_connections_test_mode
lecc    le_create_connection
leccc   le_create_connection_cancel
lecu    le_connection_update
lercprr le_remote_connection_parameter_request_reply
lercprnr        le_remote_connection_parameter_request_negative_reply
hcconn  host_controller_connection_information
scon    stream_connect
sdis    stream_transform_disconnect
```

Figure 3.2: BTCli Output for find connect Command.

3.4.1.3. Command Arguments and Parameters

Typing a question mark after a command displays the syntax for the specified command. For example e.g. `asrc ?` displays the syntax for the `accept_connection_request_and_switch` command.

Multiple syntax options may be given if a command supports variable parameter lists:

- Square brackets denote optional parameters;
- A trailing asterisk, `*`, denotes optional repetition, and
- A trailing plus + denotes optional repetition with at least one parameter required.

HCI command parameters are always named using the lowercased first letter of each part of their name (e.g. `nci` for `num_current_iac`), as given in the HCI section of the *Bluetooth Core Specifications*, except when it is necessary to distinguish `min` from `max`. For example the parameters displayed for the maximum and minimum sniff:

intervals for the `sniff_mod`e command are `smaxi` and `smini` respectively.

Commands accept textual arguments, where appropriate (this is normally the case for enumerations, and sometimes for bitmaps too); the command syntax lists the abbreviated forms of these, by following the parameter name with a colon and a pipe or vertical slash separated | list of possible textual arguments (or three dots for BCCMD/PS, where there are many possibilities).

Numeric arguments are also accepted.

Some commands accept variable numbers of arguments to support common defaults. These are:

- `inquiry` (default parameters are: `GIAC`, `maximum duration`, `unlimited number`), so the environment can be scanned using just `i`.
- `create_connection` (default parameters are: last `BD_ADDR`, `all data packets`, `R1`, `mandatory PSM`, `invalid clock offset`, `reject master-slave switch`), so a connection to a device found during inquiry can be made using just `cc`, or, for example, `cc last dm1`.

  A variant called `ccs` exists which is identical except that the default for the last parameter is to accept a master-slave switch request.
- `disconnect` (default parameters are: `last connection handle`, `user ended connection reason code`), so a disconnect can be effected using just `d`.
- `accept_connection_request` (default parameters are: `last` `BD_ADDR`, `remain as slave`), so a connection request can be accepted using just `acr`.
  A variant called `acrs` exists which is identical except that the default for the last parameter is to request a master-slave switch request.
- `reject_connection_request` (default parameters are: `last BD_ADDR`, `security reasons reason code`), so a connection request can be rejected using just `rcr`.
- `link_key_request_reply` (default parameters are: last `BD_ADDR`, `last key used` (defaults to `0x31323334353637383132333435363738`)), so a link key request can be satisfied using just `lkrr`.
- `link_key_request_negative_reply` (default parameters are: last `BD_ADDR`), so a link key request can be rejected using just `lkrnr`.
- `pin_code_request_reply` (default parameters are: last `BD_ADDR`, last PIN code used (defaults to 1234), so a PIN code request can be satisfied using just `pcrr`.
- `pin_code_request_negative_reply` (default parameters are: last `BD_ADDR`), so a PIN code request can be rejected using just `pcrnr`.
- `remote_name_request` (default parameters are: `last BD_ADDR`, `R1`, `mandatory PSM`, `invalid clock offset`), so, for example, the name of a device found during inquiry can be obtained using just `rnr`.
- `read`/`delete_stored_link_key` (default parameters are: `undefined` `BD_ADDR` if `all` is specified, one if `BD_ADDR` is specified).
- `hold_mode, sniff_mode`, `park_mode` (default parameters are: the same minimum interval as the given (maximum) interval), so, for example, a hold can be requested using just `hm l 1000`.
- `send_file` / `receive_file` (default parameter is: `last ACL connection handle`)
- `send_sco_data` (default parameter is: `last SCO connection handle`)
- `send_sco_file` (default parameters are: `last SCO connection handle`, `value of PS key PSKEY_H_HC_FC_MAX_SCO_PKT_LEN, value of PS key PSKEY_H_HC_FC_MAX_SCO_PKTS`)
- `send_acl_data` (default parameters are: `last ACL connection handle, start packet, point-to-point length corresponding to data to send plus L2CAP header, channel ID 0x0040`), so a packet including a suitable L2CAP header can be sent using just `acl "hello!"`.

All commands which take only a connection handle as a parameter use the last ACL connection handle as the default value. For example the link supervision timeout for an ACL connection which has just been established can be obtained using just `rlst`.

3.4.2. BCCMD Support

BCCMD commands are divided into read and write operation. Reads are called get requests (`bccmd_getreq`) and writes are called set requests (`bccmd_setreqs`). See *HQ and BCCMD Commands*.

3.4.2.1. bcget

`bccmd_getreqs` may be issued using `bcget`.

Table 3.2 defines special variables supported in **BTCli**. These variables are treated like ordinary variables, and are accessed through `bcget`.

Special Variable	Description
`adcres`	This variable forces an ADC measurement on one of the ADC channels. Valid ADC channels are described in the *HQ and BCCMD Commands*.
`cryptkeylen`	This variable returns the length, in bytes, of the current encryption key used on an existing ACL connection.
`piconet_instant`	This variable gives the Bluetooth clock value used at that instant on the ACL connection specified as a parameter.
`enable_afh`	This variable controls the use of Adaptive Frequency Hopping (AFH) for the ACL connection specified as a parameter.
`rssi_acl`	This variable gives the RSSI value for the ACL connection specified as a parameter.
`get_clr_evt`	This variable obtains the current value of a specified internal system event counter (as a parameter to this command). The system counter is then cleared. The counters available for reading are described in *HQ and BCCMD Commands*.
`get_next_builddef`	This variable extracts a set of identifiers that describe the major characteristics of the firmware. The latest build definitions are described in *HQ and BCCMD Commands*.

Table 3.2: Description of Special BCCMD Variable

3.4.2.2. bcset

`bccmd_setreqs` may be issued using `bcset`.

Arguments should be of appropriate sizes:

- No argument should be specified for parameterless varIDs
- An argument which fits in 8/16/32-bits should be given for varIDs which correspond to variables of size 8/16/32-bits.

3.4.2.3. bcfind

Use `bcfind` to list all BCCMD varIDs, and `bcfind "substring"` to find a varID based on part of its name.

3.4.3. Persistent Store Support

For a detailed explanation of how Persistent Store (PS) values are handled by BlueCore devices see *HQ and BCCMD Commands*.

3.4.3.1. psget

PS variables may be read using `psget`. If the stores to read are not specified, the default set is assumed. If no length is specified then `pssize` is issued as part of the command to determine the length of the data. When the `bccmd_getresp` comes back it will have been truncated to the appropriate size for the requested PS Key's value. If the `keyID` is invalid or any other error occurs, all the value elements are truncated away. Presentation keys may be read; in which case any length specified is ignored.

3.4.3.2. psset

PS variables may be written using `psset` (or `pssets`, to specify explicitly the store to be used). The length specified in the request corresponds to the number of trailing arguments given. It is advisable to use `psget` first to make sure what the correct value size is.

Note:

Each trailing argument should fit in 16-bits. Presentation keys may be written.

3.4.3.3. psclr and psclrall

Individual PS variables may be cleared using `psclr`. Alternatively, `psclrall` can be used to clear all, however, this should only be used with appropriate care.

3.4.3.4. psnext and psnextall

The next PS variable can be found using `psnext`. If no store is defined as a parameter, the default store read is the default store.

3.4.3.5. pssize

The size of a PS variable can be found using `pssize`. If the store(s) to read are not specified, the default set is assumed; if an error occurs then the `keyID/size` is not displayed.

3.4.3.6. psmt

`psmt` obtains the memory technology type being used to implement the various stores (i.e. default, `psram`, `psi`, and `psf`) as described in *HQ and BCCMD Commands*.

3.4.3.7. psslurp

The command `psslurp` is used to obtain a listing of all values that are different from the default values of the current firmware. This command is useful for support requests to CSR support, especially if the implementation in question has involved significant modification of the persistent store.

If no store type is defined as a parameter, the default store type read is the implementation (psi) store.

The output of `psslurp` is suitable for consumption by **BTCli**. The implementation store is read by default, but other stores may be listed. If more than one store is specified, then the store the PS variable was found in is not listed.

Important Note:

> This can makes restoring with guaranteed accuracy impossible (e.g. the distinction between the factory store and the implementation store is lost if both are listed simultaneously).

3.4.3.8. psfind

`psfind` is used to list all PS `keyIDs`, and `psfind "substring"'` to find a `keyID` based on part of its name.

3.4.4. Radio Test Support

A radiotest command may be requested using `rt <command> <parameters>`.

`rtfind` is used to list all radiotests, and `rtfind "substring"` to find a radiotest based on part of its name.

3.4.5. I2C Test Support

It is possible to execute I^{2}C Tx and Rx tests using the `i2c <parameters>` command. BlueCore acts as an I^{2}C master, not as an I^{2}C slave.

This **BTCli** command is described by the `I2C_Transfer` BCCMD definition in *HQ and BCCMD Commands*.

3.4.6. EEPROM Support (e2set and e2get)

For any BlueCore-ROM based hardware implementation that makes use of an EEPROM to store PS Key information, **BTCli** can be used to set the EEPROM device characteristics using the `e2set <parameters>` command.

The `e2get` command can be used to read the EEPROM device characteristics that have been previously written.

These **BTCli** commands are described in the `E2_Device` BCCMD definition in *HQ and BCCMD Commands*.

3.4.7. FM Radio Support

BTCli can be used to exercise any hardware implementation that uses a BlueCore with an integrated FM radio. The FM radio command set is described in *BlueCore-FM FM API Specification.*

3.4.8. Host Query (HQ) Support

`hqpdu_setreqs` are decoded in a limited way. The varID and status is decoded, but the payload data is simply displayed as raw hex. In the case of faults, the fault ID is decoded as well.

For more details of HQ, see the *HQ and BCCMD Commands*.

3.4.9. VM Channel Support

BTCli provides rudimentary support for the VM channel (BCSP channel 13) used in BlueLab.

If a packet is received, then it is displayed as raw hex octets if it is not a `VmPutChar` packet. `VmPutChar` packets are buffered up for a given sub-channel until newline or zero byte (null character) is received, at which point the buffered data is printed as a string.

Packets can be sent with the `vm_data` command, supplying a sub-channel and an optional string.

3.4.9.1. Raw Commands (Raw Byte Sequences)

Raw HCI commands can be sent using raw.

Note:

> This bypasses HCI flow control.

Multi-octet arguments can be specified as a hex string where the size depends on the number of digits given. Thus, `0x123` is 2 octets while `0x000123` is 3 octets. Multi-octet arguments are transmitted Least Significant Bit (LSB) first.

3.4.10. Predefined Command Shortcuts

BTCli provides a few quick command short hands which are useful for common activities.

- `slave` sets an event filter to auto-accept connections, enables page and inquiry scanning and reads the Bluetooth Device Address. This is equivalent to `sef cs all` a then `wse ip` and then `rba`, see example in section 5.1.
- `msp` tells the BlueCore to send SCO data for the next connection to the Pulse Code Modulation (PCM) hardware. This is equivalent to `bcset map_sco_pcm 1` command.
- `settran` can be used to set the PS Keys for transport and/or baud rate. If the transport is set, the UART configuration appropriate to it is automatically set.

Note:

> PS Key USB_PIO_PULLUP is not automatically set and should be set manually if needed.

- `ping` can be used to send an ACL packet with a timestamp.
- `aclab` and `aclpb` can be used to send active/piconet broadcast L2CAP messages.

3.5. BTCli Parameter Considerations

3.5.1. Numerical Parameters

Whenever a number is given, it is taken to be entered most significant part first i.e. if a link key is given as `0x1245` the LSB of the link key will be `0x45` (the MSB being `0x12`).

Whenever a string is given, the first character is taken as the LSB.

3.5.2. String Parameters

Strings are not taken to include a NULL terminator unless explicitly specified using an escape sequence.

Strings are delimited by " (quote) characters.

The following sequences should be used for special characters:

- `\r, \n, \t, \0` (13, 10, 9, 0)
- `\xdd`, where `d` is a hex digit i.e. two hex digits are required
- `^@, ^A ... ^Z, ^[, ^\, ^], ^^, ^_` (0-31)
- `^?` (127)
- `\\` ('\')
- `\"` ('"')
- `\^` ('^')

These sequences are used on output.

Note:

> The first two sets of escape sequences listed above are not recognized in strings corresponding to filenames or commands.

This allows `\` directory separators in (Windows) filenames to be used without needing to be escaped (as `\\`).

3.5.3. Parameter Size Considerations

Arguments for parameters less than 33-bits in length can be given in hex, decimal, octal or binary, by prefixing with `0x`, nothing, `0` or `0b` respectively. Arguments for parameters of size greater than 32-bits must be given in hex (or as strings, where supported).

If an argument is given which is too big for a given parameter, the results are undefined (typically the command will be rejected as if it had been mistyped).

3.6. ACL Data

ACL packets can be sent using `acl`; the command `acl ?` gives the syntax. The parameters are abbreviated as per HCI section of the *Bluetooth Core Specification*.

An L2CAP header is normally generated for start packets; the only way to send an ACL start packet without an L2CAP header is to use the form of this command where the `Packet_Bounday_Flag` is explicitly specified but no L2CAP length or channel ID are specified (e.g. `acl acl0 s ptp "x"`).

3.7. SCO Data

SCO packets can be sent using `sco "string"`, with an optional preceding connection handle. If no connection handle is specified the last SCO connection handle is used.

Files of SCO data can be sent using sends `"filename"`, with an optional preceding connection handle. SCO data files can be quickly generated if the SCO data from a previous test is routed to a SCO log file as described in Table 3.1.

The data rate is 8000 samples per second where a sample is 8 or 16-bits in size depending on the current voice settings. The default SCO packet data size is 48 samples per SCO packet; this may optionally be changed by specifying a SCO packet payload size in octets after the filename.

It is not possible to abort sending a SCO data file, and it is not possible to do anything else while sending a SCO data file. In particular, this means it is not possible to send more than one SCO data file at the same time.

By default, 40 samples are pre-queued to guard against underrun. It is possible to select the number of samples to pre-queue by setting the **BTCli** or environment variable `SCOPREQUEUE`. Windows environment variables are adjusted in the **Systems Properties** window, which is accessible from the **Control Panel**. The **Advanced** tab of the **System Properties** window allows access to changing environment variables.

Note:

> Sending SCO over HCI involves fairly stringent real-time requirements, such as a minimum UART baud rate of 230400 bps. Information regarding the extent to which these real-time requirements were violated is printed when the SCO data file has been sent. Parameters that may influence the real-time performance of the system include the machine loading, the process priority, and the SCO packet payload size.

Files of SCO data can be received using a command line argument, as described in section 3.3.

3.7.1.1. Special Symbols

Wherever a Bluetooth device address is needed the last Bluetooth device address can be used by entering last (or just `l`).

The last Bluetooth device address is updated:

- From the **last** result in an `inquiry_result` event
- From a successful `connection_complete` event
- From a `connection_request` event
- From a `link_key_request` event
- From a `pin_code_request` event
- From a `create_connection` command
- From a successful `synchronous_connection_complete` event

last can also be used for ACL and SCO connection handles.

Note:

> In the `setup_synchronous_sonnection`, `change_connection_packet_type` and `disconnect` commands the **last** SCO handle, if valid, is chosen; otherwise the **last** ACL handle is always used.

The **last** ACL connection handle is updated:

- From a successful `connection_complete` event if the link type is ACL
- From an ACL packet (transmission (tx) or reception (rx)), if not on a broadcast handle
- From a successful `disconnection_complete` if the handles match
- From a `command_complete` (Reset)

The **last** SCO connection handle is updated:

- From a successful `connection_complete` event if the link type is SCO
- From a successful `synchronous_connection_complete` event
- From a SCO packet transmission
- From a successful `disconnection_complete` if the handles match
- From a `command_complete` (Reset)

Note:

> **last** handles do not nest. For instance, after an ACL `disconnection_complete` the previous ACL connection handle becomes invalid.

ACL and SCO connection handles are maintained symbolically in symbols called `acl0` to `acl9` and `sco0` to `sco9`. These can be shortened to `a0-a9` and `s0-s9` respectively. The lowest-numbered free symbol is used when one is allocated. The symbols are updated:

- From a successful `connection_complete` event
- From a successful `synchronous_connection_complete` event
- From a successful `disconnection_complete` event
- From a `command_complete` (Reset)

Active and piconet broadcast handles are maintained symbolically in symbols called `ab` and `pb`. These are currently hard-wired, which means that if the host controller uses them in a `connection_complete` event the situation cannot be untangled as described in the HCI section of the *Bluetooth Core Specification*. The values have been chosen such that this problem cannot occur with BlueCore, but this problem may occur with third-party host controllers.

The results are undefined when a last or handle symbol is used but no such entity exists, either because no such entity ever existed or because it has ceased to exist. Typically, the command will be rejected as if it had been mistyped.

A list of **last** and handle symbols can be obtained using sym.

3.8. Event Decoding

Full textual forms are always used for decoding events. This includes command names in `command_status` and `command_complete` events, where appropriate, and where the event parameter has a valid value. Where reasonable, the form as given in the HCI section of the Bluetooth Core Specifications is used (lowercased).

Where numeric forms are used, the number is output in hex (with a number of digits corresponding to the parameter size), preceded with the parameter name, using the same naming convention as for command parameters (e.g. `nhcp:0x01` sets `num_hci_command_packets` to `1`).

Some events are decoded in a special way:

- Supported features (resulting from a `read_local/remote_supported_features` command) are decoded numerically, followed by a parenthesised pipe or vertical slash separated "`|`" (OR) list of features, where a parenthesised tilde "`~`" (NOT) denotes an unsupported feature.
- Link policy settings (resulting from a `read_link_policy_settings` command) are decoded numerically, followed by a parenthesised pipe or vertical slash separated "`|`" (OR) list of features, where a tilte parenthesised "`~`" (NOT) denotes a disabled policy. The flow control lag is decoded numerically.
- `loopback_command` starts `loopback_command hcp:` and then prints out the octets of the `hci_command_packet` in groups of five, separated by a hyphen `-`.

4. Advanced Topics

4.1. Using Variables

Variables are substituted on input lines. A variable name consists of a string of the alphanumerics plus underscore. A variable is (re)defined using `set <name>=<value>`; it is substituted using `$<name>`.

Note:

> Substitution occurs without regard to strings. If a variable has not been defined then it is looked up as an environment variable. Non-existent variables substitute to nothing. To enter a single `$` escape it with another.

To delete a variable use `set <name>=`; to list a variable use `set <name>`; to list all variables use `set`. `echo` can be used to see what a given input line expands to.

The text to the right of the `=` is taken as-is, including any leading or trailing whitespace. To use a variable immediately followed by non-whitespace use a trailing `$` and whitespace. This degree of control is unlikely to be required.

Note:

> Variables are distinct from symbols as described in section 3.7.1.1 of this document.

Table 4.1 shows some example command line inputs and the resultant contents of the variable.

Command Line Entry	Resultant Variable Content

`‘set foo=bar’`
`‘$foo’`
	`‘bar’`

`‘$$foo’`	`‘$foo’`
`‘$foo spong’`	`‘bar spong’`
`‘$foo$ spong’`	`‘barspong’`
`‘$foo$spong’`	`‘bar’`

`‘set foo=babar’`
`‘$foo’`
	`‘babar’`

`‘set foo= babar’`
`‘spong$foo’`
	`‘spong babar’`

`‘spong $foo’`	`‘spong  babar’`

`‘set foo=’`
`‘$foo bar’`
	`‘bar’`

`‘set foo= ‘`
`$foo bar’`
	`‘  bar’`

Table 4.1: Command Line Examples

4.2. Scripting

Scripts can be launched with `exec` or can be executed upon start if the command line option `-x` is used to specify a script. It is important to note that some commands and events may still be in transit even after **BTCli** has started up and provides a user prompt. It is best to take care before entering more commands to ensure all the script activity has completed.

Scripts may be nested, meaning it is possible to execute a script (`exec`) from within another script.

Lines starting with # are comment lines and are skipped.

The sleep command can be used in scripts to pause for a specified number of milliseconds. Additionally, sleep can be used to regulate script flow. The default time for a sleep command (i.e. no parameter is given) is 1000 milliseconds.

Note:

> HCI flow control is automatically enforced when using **BTCli**, this means a series of commands can be issued in a script and they will be passed to the host controller when the host controller declares itself ready to accept them through the `nhci` parameter of the `command_status` and `command_complete` events. The exceptions are the following HCI commands:
>
> - `link_key_request_reply`
> - `link_key_request_negative_reply`
> - `pin_code_request_reply`
> - `pin_code_request_negative_reply`
> - `reset`
> - `host_number_of_completed_packets`
> - The HCI extension command for tunnelling (`0xfc00`)

BTCli also enforces BCCMD flow control; this means that a series of BCCMDs, including `psgets` and `pssets`, can be issued in a script and they will be passed to the host controller when it has responded to the previous BCCMD. The exceptions are for VM data and for the following BCCMDs:

- `cold_reset`
- `warm_reset`

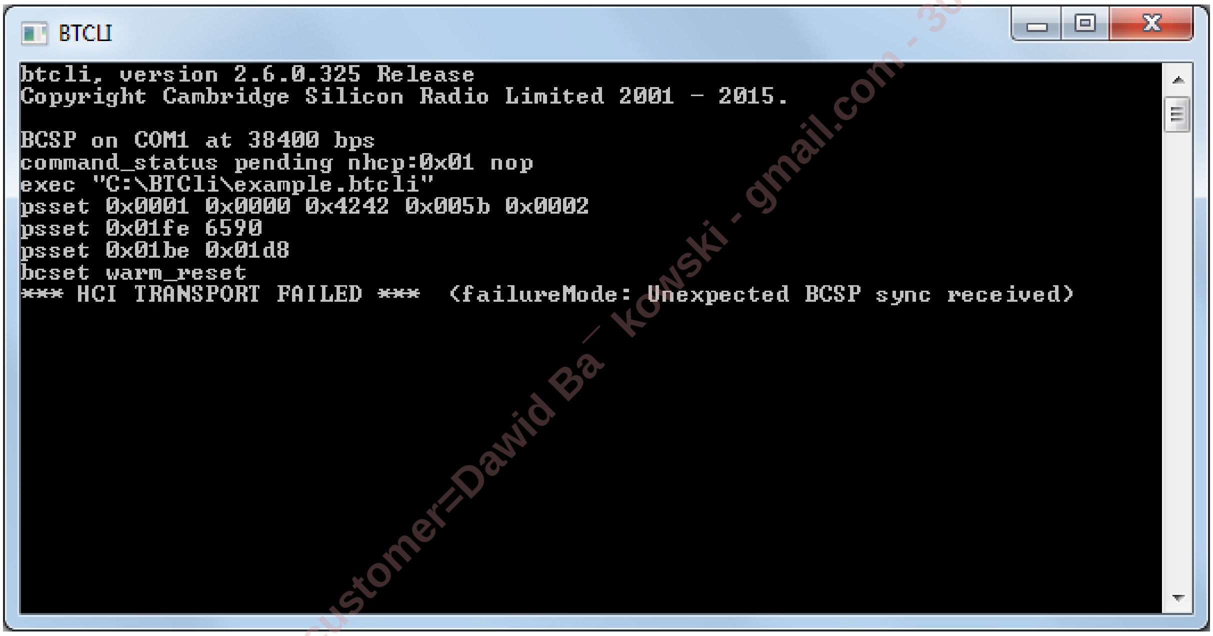
4.3. Bootstrapping BlueCore ROM Devices

Bootstrapping a BlueCore ROM device is mostly an extension or implementation of scripting. To bootstrap a ROM device, it is best to create a script with a sequence of `psset` commands in which all the necessary PS Keys are set to the appropriate values. Alternatively, hand typing each command works as well, but is less efficient.

A short example of a bootstrap script file might include at least the following lines:

```
# Set the Bluetooth device address to 0x00025b004242
psset 0x0001 0x0000 0x4242 0x005b 0x0002
# Set the crystal frequency to 26MHz
psset 0x01fe 6590
# Set the UART baud rate to 115kbps
psset 0x01be 0x01d8
# Minimal config is set, warm reset so the settings take effect
bcset warm_reset
```

Figure 4.1 shows the results of launching **BTCli** and manually executing the above example script.



```
btcli, version 2.6.0.325 Release
Copyright Cambridge Silicon Radio Limited 2001 - 2015.

BCSP on COM1 at 38400 bps
command_status pending nhcp:0x01 nop
exec "C:\BTCli\example.btcli"
psset 0x0001 0x0000 0x4242 0x005b 0x0002
psset 0x01fe 6590
psset 0x01be 0x01d8
bcset warm_reset
*** HCI TRANSPORT FAILED ***  (failureMode: Unexpected BCSP sync received)
```

Figure 4.1: Example BlueCore ROM Bootstrap with BTCli

Notes:

BTCli was launched COM1, BCSP, at 38.4kbps. This enabled the BlueCore ROM device to auto-baud and to connect with **BTCli**.

After the script was run, the baud rate was changed to 115kbps, which is not the same baud rate **BTCli** was launched with initially. The `*** HCI TRANSPORT FAILED ***` message indicates **BTCli** lost sync with the BlueCore.

At this point, it is necessary to close this **BTCli** window and to reopen a new window specifying 115kbps as the baud rate, to allow a connection to the ROM device for testing.

If, however, the bootstrap resulted in using the same baud rate that **BTCli** was opened with originally, it is possible to type `restart` to cause **BTCli** to reset itself (not BlueCore) and restart its host interface. If BCSP or H5 was used, then the appropriate link establishment restarts. Other host interfaces would be restarted cleanly as well.

4.4. Throughput Testing (Sending Files)

Files can be sent/received using send `"filename"` and `recv "filename"` with an optional preceding connection handle, otherwise the last ACL connection handle is used. Files should be less than 2GB in size.

`recv` should be started first. Multiple sends and receives can be performed simultaneously. All the sends must use different connection handles, and all the receives must use different connection handles.

It is permissible to send non-file data on the same connection handle as one currently used to receive a file, but this should honour L2CAP semantics or at least start with a start packet. Additionally, it should not use L2CAP channels with ID `0x0041` or `0x0042`, which are the channel IDs **BTCli** uses for its file transfers.

`number_of_completed_packets` events are suppressed on the sending side during a file transfer, if they entirely pertain to connection handles currently used for sending.

A send or receive can be aborted by using `send` or `recv` without a filename. Sends and receives are automatically aborted if a disconnection occurs for the connection handle they are on.

Aborting a send will result in a few `number_of_completed_packets` events appearing on the aborting side.

The data transfer rate is printed at the end of a receive event. The data transfer rate to date during a receive can be obtained using `fstat`.

Note:

All rates computed are net rates.

The file transfer protocol is as follows.

4.4.1. File Transfer Protocol

- L2CAP semantics are honoured.
- Data is transferred in L2CAP messages.
- The L2CAP message size is variable.
- An L2CAP message is carried in the payload of an HCI ACL data start packet and zero or more HCI ACL data continuation packets.
- L2CAP messages start with an L2CAP header containing the L2CAP payload size (2 octets) and the channel ID (2 octets).
- A file send consists of transmission from file sender to file receiver of a ‘start’ L2CAP message, zero or more ‘data’ L2CAP messages and a ‘finish’ L2CAP message.
- If the `recv` command has been used by the file receiving session, a ‘finish acknowledgement’ L2CAP is transmitted from file receiver to file sender after the ‘finish’ L2CAP message.
- Packets may not be broadcast.

The format of a ‘start’ L2CAP message is:

- L2CAP channel ID 0x0041 is used.
- L2CAP payload: octet 1: state indicator (0x00) representing ‘start’.
- L2CAP payload: octet 2- octet 5: size of the file name (including path).
- L2CAP payload: octets 6 and following: name of the file.

The format of a ‘data’ L2CAP message is:

- L2CAP channel ID 0x0042 is used.
- L2CAP payload: octets 1 and following: contents of the file

The format of a ‘finish’ L2CAP message is:

- L2CAP channel ID 0x0041 is used.
- L2CAP payload: octet 1: state indicator (0x04) representing ‘finish’.

- L2CAP payload: octet 2 – octet 5: size of the file transferred

The format of a ‘finish acknowledgement’ L2CAP is:

- L2CAP channel ID 0x0041 is used.
- L2CAP payload: octet 1: state indicator (0x05) representing ‘finish acknowledgement’.

By default, a large L2CAP message size is chosen, for optimum throughput. It is possible to select the L2CAP message size by setting the **BTCli** or environment variable `SENDL2CAPSIZE`. The value must be between 5 and 65535 inclusive. Windows environment variables are adjusted in the **Systems Properties** window, which is accessible from the Control Panel. The **Advanced** tab of the **System Properties** window allows access to changing environment variables.

If host controller to host flow control is enabled, then a `host_number_of_completed_packets` command is automatically sent for every packet received during a file transfer (including those pertaining to non-file data).

Note:

If missing octets are detected during reception, the missing octets are written as `0xd0`.

4.5. Runtime Options

Run-time options may be selected using `opt`, with an argument consisting of + (to enable) or - (to disable) followed by:

- `acl` to select whether to display incoming ACL packets like the -a command-line option. (defaults to enabled).
- `sco` to select whether to display incoming SCO packets like the -s command-line option. (defaults to enabled).
- `cqddr` to select whether to display LM CQDDR debug data.
- `power` to select whether to display LMP power messages.
- `time` to select whether to display timestamps (like the -t command-line option).
- `awlps` to select whether to automatically issue a `wlps l eall` after (ACL) connection creation like the -w command-line option.
- `tag` to select whether to display a tag at the start of each line this should typically have a trailing space but no leading space, e.g. `opt +tagMASTER`.
- `ascii` to select whether data returned by BCCMD can be displayed as an ASCII string. If the option is selected, the program will try to decide whether the data, or an initial subset of the data up to a zero value, is printable.
- `raw` to select whether to display the octets transmitted and received over the HCI transport. In the case of USB received data, the USB endpoint type is prepended (within braces) - one of `ctl` (control), `blk` (bulk), `isc` (isochronous) and `int` (interrupt) . For example, `hci_rx(int) <received data>` .The `rawlog` command can be used instead, if this option is disabled, to display the octets transmitted and received since the last time the `rawlog` command was used or the `raw` option was disabled. The `rawlog` command does not prepend the output with the USB endpoint type.
- `c2` as `cqddr` second, but only affects certain packet types.
- `expand` to show variable substitution expansion. See section 4.1 for information on variables.
- `afh` to show AFH packets (defaults to enabled).
- `fmrssi` to pretty-print FM RSSI packets.
- `fp` to show FastPipe packets (defaults to enabled).
- `scolog<n>` to turn logging on for SCO channel `<n>. +scolog<n>` requires a filename to follow. The channel number defaults to 0. Use of `-scologall` is also allowed.
- `log` to turn logging on or off. `+log` requires a filename to follow.
- `gps` to show GPS packets
- `eir` to decode EIR data structures (defaults to enabled)
- `rl` turn raw logging on or off. `+rl` requires a filename to follow.
- `auto_rawlog` to automatically do a `rawlog` if the HCI transport fails.
- `recvaclhex` to display received acl data in hexadecimal format, rather than character format. In hexadecimal format, the octet order is reversed relative to character format, in order to match the display format used when sending hexadecimal data.
- `blockresetrestart` to block commands `bcset warm_reset`, `bcset cold_reset` and `restart` until all prior commands have been executed. The default behaviour for these commands is not to wait for any prior queued commands to execute.

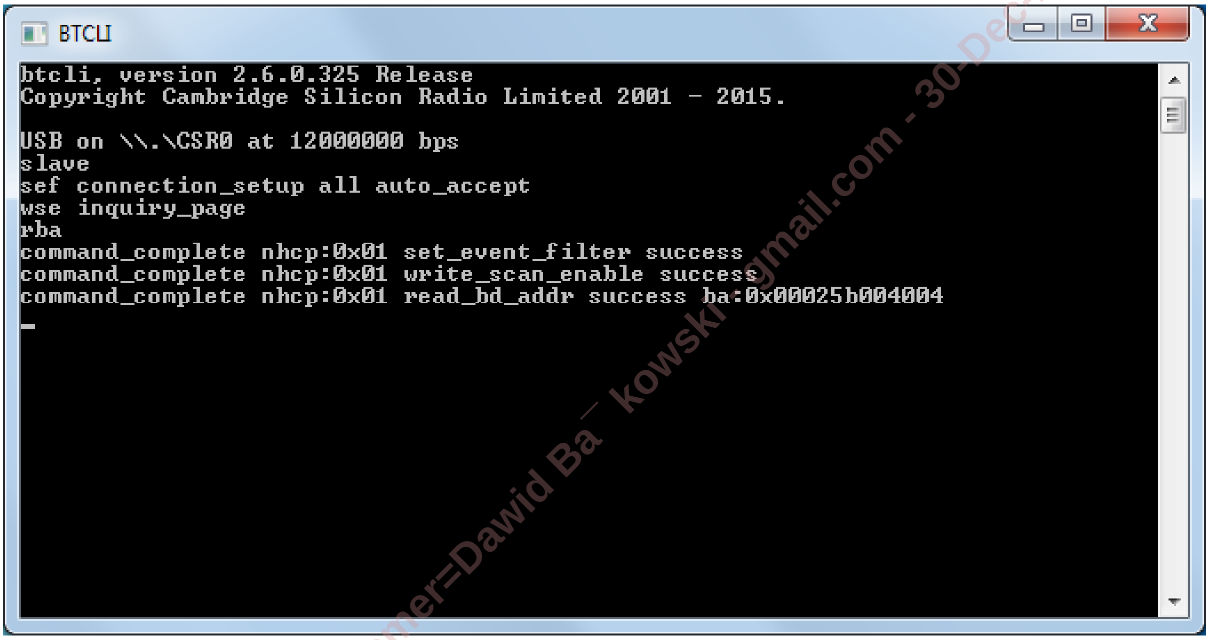
Options default to disabled unless otherwise noted above or unless countermanded on the command line.

5. Example Usage

This section provides a few examples of common scenarios using **BTCli**. Using these examples, one should be able to extrapolate how to perform other tests and operations within **BTCli**.

5.1. Example Connection Sequence

Figure 5.1 shows the output of the **BTCli** window when the slave device is set up using the built-in **BTCli** command slave. This command automatically sets the device into discoverable and connectable mode (inquiry scan and page scan enabled (`wse inquiry_page`), and it enables the event filters to automatically accept incoming connections (`sef connection_setup all auto_accept`). Finally the script reads the local device's Bluetooth device address (`rba`) to make it easier to then type that address into master device's **BTCli** window in the create connection command.

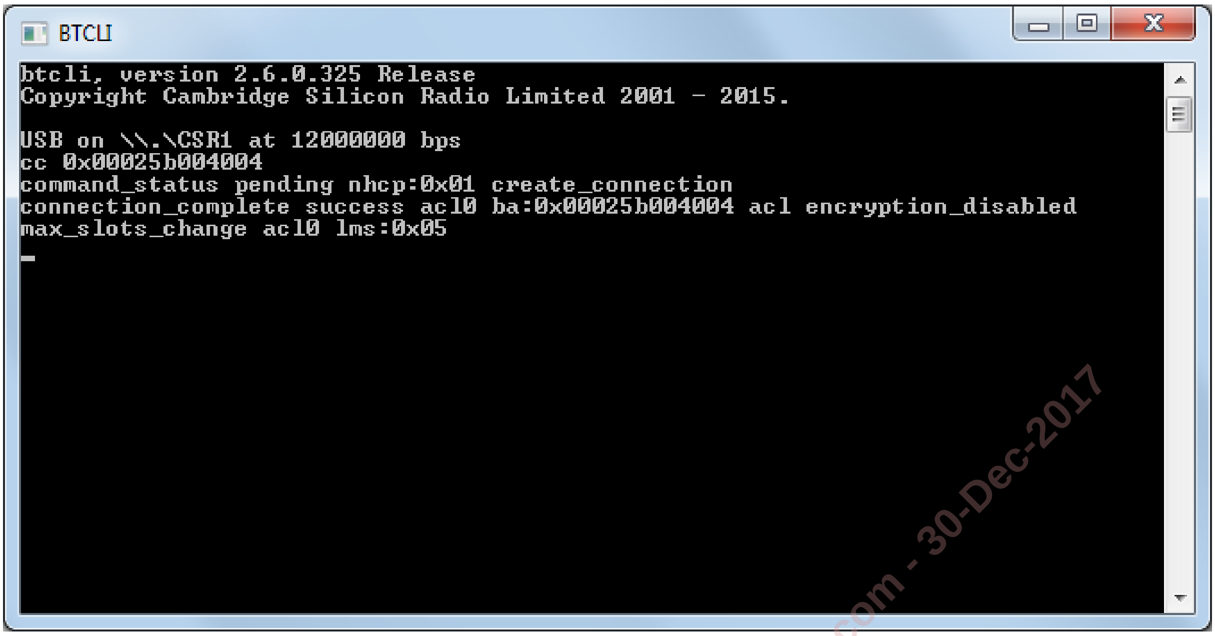


```
btcli, version 2.6.0.325 Release
Copyright Cambridge Silicon Radio Limited 2001 - 2015.

USB on \\.\CSR0 at 12000000 bps
slave
sef connection_setup all auto_accept
wse inquiry_page
rba
command_complete nhcp:0x01 set_event_filter success
command_complete nhcp:0x01 write_scan_enable success
command_complete nhcp:0x01 read_bd_addr success ba:0x00025b004004
_
```

Figure 5.1: BTCli Output For Slave Command

Figure 5.2 shows the output of the **BTCli** window for the master device creating a connection to the slave device from Figure 5.1. In this window, the command `cc 0x00025b004004` was typed.

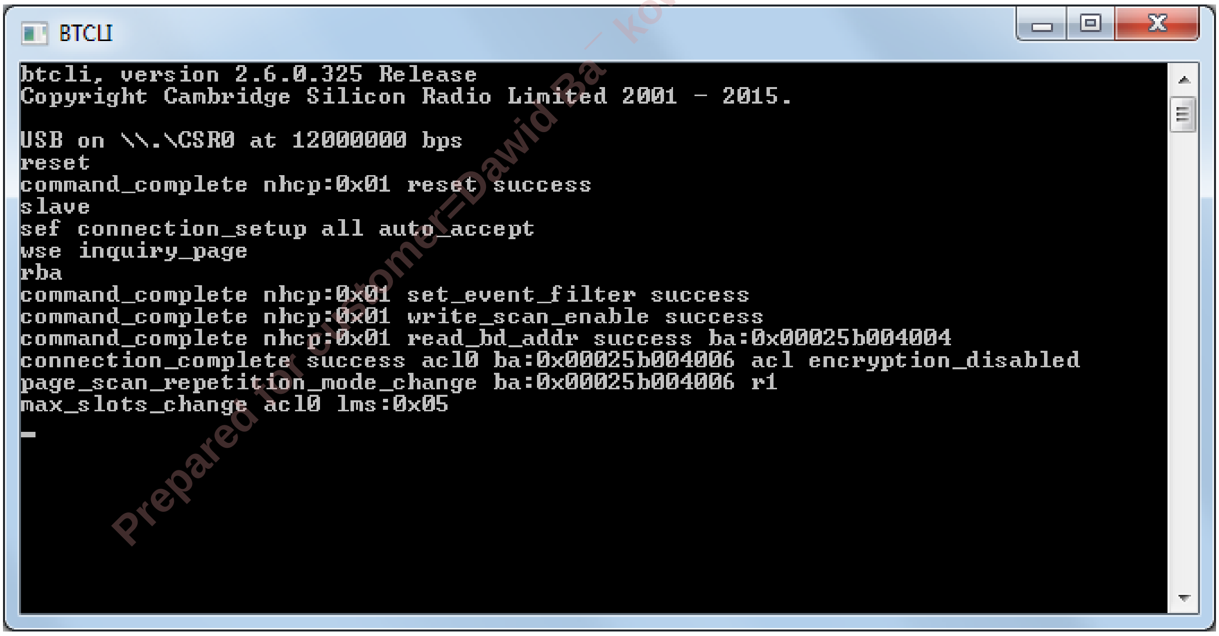


```
btcli, version 2.6.0.325 Release
Copyright Cambridge Silicon Radio Limited 2001 - 2015.

USB on \\.\CSR1 at 12000000 bps
cc 0x00025b004004
command_status pending nhcp:0x01 create_connection
connection_complete success acl0 ba:0x00025b004004 acl encryption_disabled
max_slots_change acl0 lms:0x05
_
```

Figure 5.2: BTCli Output Master Device Create Connection Command

Figure 5.3 shows the resulting output from the slave device as the incoming connection is accepted. Only the command slave was typed. The additional text is printed by **BTCli** as an indication of the incoming connection status.



```
btcli, version 2.6.0.325 Release
Copyright Cambridge Silicon Radio Limited 2001 - 2015.

USB on \\.\CSR0 at 12000000 bps
reset
command_complete nhcp:0x01 reset success
slave
sef connection_setup all auto_accept
wse inquiry_page
rba
command_complete nhcp:0x01 set_event_filter success
command_complete nhcp:0x01 write_scan_enable success
command_complete nhcp:0x01 read_bd_addr success ba:0x00025b004004
connection_complete success acl0 ba:0x00025b004006 acl encryption_disabled
page_scan_repetition_mode_change ba:0x00025b004006 r1
max_slots_change acl0 lms:0x05
_
```

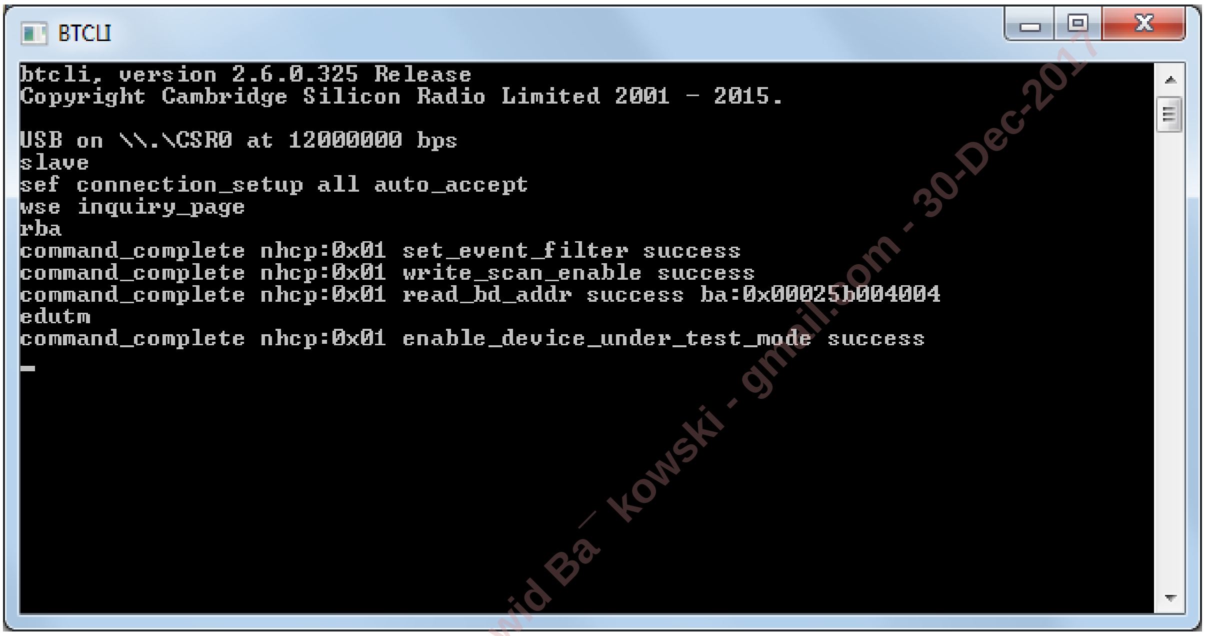
Figure 5.3: BTCli Output For Slave Device Incoming Connection

5.2. Placing a Device into HCI Device Under Test Mode

Placing a device into the Bluetooth device under test (DUT) mode is a common use of BTCli and often causes confusion. There are only two commands involved:

1. `slave`: See section 5.1,
2. `edutm`: To enable device under test mode.

Figure 5.4 shows the output of **BTCli** after placing a device into discoverable, connectable, and test mode.



```
btcli, version 2.6.0.325 Release
Copyright Cambridge Silicon Radio Limited 2001 - 2015.

USB on \\.\CSR0 at 12000000 bps
slave
sef connection_setup all auto_accept
wse inquiry_page
rba
command_complete nhcp:0x01 set_event_filter success
command_complete nhcp:0x01 write_scan_enable success
command_complete nhcp:0x01 read_bd_addr success ba:0x00025b004004
edutm
command_complete nhcp:0x01 enable_device_under_test_mode success
_
```

Figure 5.4: BTCli Output for Slave Device Placed Into DUT Mode

Document References

Document	Reference
HQ and BCCMD Commands	CS-227432-SP
Casira User Guide	CS-102077-UG
BlueCore Serial Protocol (BCSP)	CS-101685-SP
BlueCore-FM FM API Specification	CS-101761-SP
Bluetooth Core Specifications	www.bluetooth.org

Terms and Definitions

ADC	Analogue-to-Digital Converter
AFH	Adaptive Frequency Hopping
BCCMD	BlueCore Command (see CS-227432-SP)
BCSP	BlueCore Serial Protocol
BlueCore®	Group term for CSR’s range of Bluetooth wireless technology chips
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
CRC	Cyclic Redundancy Check
CSR	Cambridge Silicon Radio
Curator	A device-management component of HydraCore devices
DUT	Device Under test
e.g.	*exempli gratia*, for example
etc	*et cetera*, and the rest, and so forth
HCI	Host Controller Interface
hif	HydraCore Host Interface, e.g. BCSP, SDIO
HQ	Host Query
HydraCore	Group term for CSR’s range of wireless multi-technology chips
i.e.	*id est*, that is
LSB	Least Significant Bit
MSB	Most Significant Bit
PC	Personal Computer
Persistent Store	Store of configuration values in non-volatile memory
PS	Persistent Store
PTAP	Packet Tap – an interface to Host Transports used by HydraCore devices
PTS	Profile Testing Suite
ROM	Read Only Memory
RSSI	Received Signal Strength Indication
Rx	Receive
SCO	Synchronous Connection-Oriented
SDIO	Secure Digital Input Output - a standard interface for input or output devices

Tx	Transmit
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
varID	A BCCMD field holding the identifier of the variable being manipulated by the command
VM	Virtual Machine