Agilent E2940A Opt. 320 C-API/PPR

Reference

Agilent Technologies

Important Notice

The information in this manual is subject to change without notice. Agilent Technologies makes no warranty of any kind with regard to this manual, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Agilent Technologies shall not be liable for errors contained herein or direct, indirect, special, incidental, or consequential damages in connection with the furnishing, performance, or use of this manual.

Authors: Stephan Greisinger and Anja Schauer, t3 medien GmbH

Contents

Introduction

The C_API Reference describes all C functions, types and definitions of the application programming interface of the Agilent PCI testcard. It also provides the commands and abbreviations to be used in the command line interface (CLI) of the graphical user interface.

To develop C programs or to use the command line interface, you should have good background knowledge of the Agilent PCI testcard and the programming models.

This is provided in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

After you have read this Guides, you will be ready to use the C-API functions/CLI commands.

The documentation of C functions needs a consistent approach to name constants, types, and functions as shown in *"Conventions" on page 14*.

For information about the structure of the reference and the structure within individual sections, see *"Sections of the C-API Reference" on page 15*.

Conventions

Programming Interfaces

The Agilent PCI testcard provides two programming interfaces:

- The application programming interface (C-API) allows control of the software by means of C function calls.
- The Command Line Interface (CLI) provides a means of using the function calls directly from the command line of the graphical user interface (GUI). No C compiler is needed. This interface is used for simple interactive testing.

The functions are described with their syntax, return value and parameters. The CLI equivalent is specified for each function. For typing convenience, an abbreviated form of each CLI command and its associated parameters is used.

Naming Conventions

The following conventions are used to name constants, types, and functions:

- Naming of Constants:

  Constants are written in capital letters. Each name begins with "B_".

  Example: B_RESLOCK_EXERCISER.

NOTE Constants which are used exclusively in Protocol Permutator and Randomizer functions begin with "BPPR_".

Example: BPPR_GEN_BUSSPEED

- Naming of Types:

  Type names are written in lower case letters. Each name begins with "b_" and ends with "type".

  Example: b_resourcetype

NOTE Types which are used exclusive in Protocol Permutator and Randomizer functions begin with "bppr_".

Example: bppr_genproptype

- Naming of Functions:

  Function names are written in lower case letters and capital letters. Each name begins with “Best”. The action is indicated by the last words in the call.

  Example: BestResourceLock

NOTE Protocol Permutator and Randomizer functions begin with “BestPpr”.

Example: BestPprGenPropSet

Sections of the C-API Reference

The C-API Reference is divided into the following sections:

- *“General Functions” on page 17*
- *“PCI Analyzer Functions” on page 43*
- *“PCI Exerciser Functions” on page 99*
- *“Interface Control Functions” on page 173*
- *“Protocol Permutator and Randomizer Functions” on page 187*
- *“Error Handling” on page 223*
- *“Type Definitions” on page 227*

All sections except the type definition section are again thematically grouped. Every group starts with an overview of the functions in logical order, followed by the full description of the functions in alphabetical order.

The type definition section is not thematically grouped. All available type definitions are listed in alphabetical order.

General Functions

The general functions are divided into the following sections:

- *"Connection and Initialization Functions" on page 17*
- *"Administration Functions" on page 23*
- *"Power Up and Reset Control Functions" on page 32*
- *"Card Status Functions" on page 39*

Connection and Initialization Functions

The following functions are used for connection and initialization purposes:

Function	Result
"BestDevIdentifierGet" on page 19	Returns the identifier of a PCI device.
"BestOpen" on page 21	Opens a connection to the testcard.
"BestClose" on page 18	Closes the connection to the testcard.
"BestPing" on page 22	Checks a connection to the testcard.
"BestRS232BaudRateSet" on page 22	Sets the baud rate if the serial interface is used.

How to use the functions is described in *"Connection and Initialization"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestClose

Call `b_errtype BestClose( b_handletype handle );`

Description Closes the session and frees any allocated memory. If the serial port has been used, it also resets the baud rate to 9600.

CLI Equivalent No CLI equivalent. Closing the CLI window executes BestClose automatically.

CLI Abbreviation No CLI abbreviation.

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestOpen" on page 21*

BestDevIdentifierGet

Call

```
b_errtype BestDevIdentifierGet(
      b_int32     vendor_id,
      b_int32     device_id,
      b_int32     subsys_id,
      b_int32     *devid );
```

Description Used if the PCI port is applied for in-system analysis (when the software is running on the system under test). The function returns the device identifier of the testcard on the PCI bus. The returned device identifier can be used as port number for the function *"BestOpen" on page 21*.

If multiple cards are plugged into the system, the number stored in the subsystem id register in the configuration space can be used to distinguish between different testcards.

NOTE This function can only be used in systems with standard PCI BIOS. For other systems, you must build the device identifier on your own. Then you must follow the rules shown in *"Device Identifier Format" on page 20*.

CLI Equivalent

```
BestDevIdentifierGet vendor_id=<vendor_id> device_id=<device_id>
subsys_id=<subsys_id>
```

CLI Abbreviation

```
diget vendor=<vendor_id> dev=<device_id> subsys=<subsys_id>
```

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

vendor_id Vendor ID of the testcard (default = 103C\h for Agilent).

device_id Device ID (vendor dependent) for the testcard (for example, 2940\h).

number Index to distinguish between different testcards in one system. The first card has index "0", the second "1", and so forth.

Output Parameters

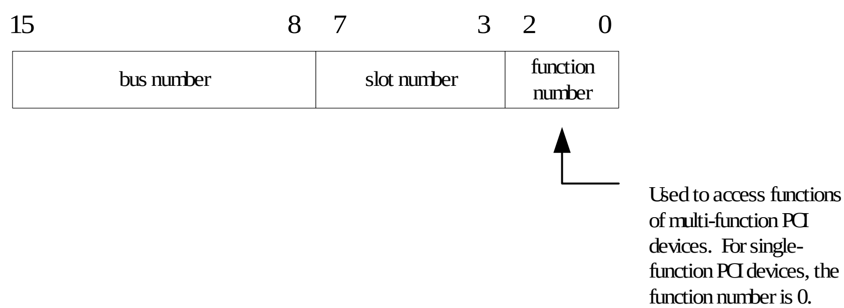
devid Device identifier within the system. This is the device number used to access the card's configuration space (for example, in *"BestOpen" on page 21*).

See also –

Device Identifier Format

Usually you will not need the device identifier format, because you only need to pass the device identifier returned by BestDevIdentifierGet to the BestOpen call.

The function co-operates only with standard PCI BIOS. If you use another BIOS or run a system without BIOS, you can build the device identifier yourself by observing the following format rules:



BestOpen

Call

```
b_errtype BestOpen(
      b_handletype *handle,
      b_porttype   port,
      b_int32      portnum );
```

Description Opens and checks the connection to the PCI testcard and initializes the internal structures and variables for the control port.

This function must be called before calling any other function. It returns a handle for the session, which is used by all subsequent C-API functions. The handle identifies each testcard and declares the control port for the session (for example, the Fast Host Interface).

You can open multiple sessions for one testcard (for example, one Fast Host Interface session and one PCI session).

NOTE You cannot open multiple sessions for the same port simultaneously.

CLI Equivalent No CLI equivalent.

The function is automatically called when you open the CLI window. Port and port name are then constant during the entire CLI session.

CLI Abbreviation No CLI abbreviation.

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

port Type of control port (for example, Fast Host Interface); see *"b_porttype" on page 261*.

portnum Control port; see *"b_porttype" on page 261*.

Output Parameters

handle Handle to identify the session (comparable to a file handle).

See also

"BestClose" on page 18
"BestDevIdentifierGet" on page 19

BestPing

Call `b_errtype BestPing( b_handletype handle );`

Description Checks the connection to the card. If the connection is ok, both card LEDs will flash simultaneously.

CLI Equivalent `BestPing`

CLI Abbreviation `ping`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle that identifies the session and the connection used by the session.

See also –

BestRS232BaudRateSet

Call

```
b_errtype BestRS232BaudRateSet(
      b_handletype   handle,
      b_int32        baudrate );
```

Description Changes the baud rate.

CLI Equivalent `BestRS232BaudRateSet baudrate=<baudrate>`

CLI Abbreviation `brset baud=<baudrate>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

baudrate Baud rate value; see table below.

See also –

Baud Rate Values

Value (CLI Abbreviation)	Baud Rate
B_BD_9600 (9600)	9600 baud
B_BD_19200 (19200)	19200 baud
B_BD_38400 (38400)	38400 baud
B_BD_57600 (57600)	57600 baud

Administration Functions

The following functions are used to get information about the system under test and to lock and unlock resources:

Function	Result
"BestVersionGet" on page 31	Checks the version of a component of the card.
"BestCapabilityCheck" on page 24	Checks if a card capability is enabled.
"BestCapabilityRead" on page 26	Reads the OR-combined capability code.
"BestSystemInfoGet" on page 30	Reads out PCI system information.
"BestResourceLock" on page 28	Locks a resource.
"BestResourceIsLocked" on page 27	Checks whether a resource is locked.
"BestResourceUnlock" on page 29	Unlocks a resource.
"BestAllResourceUnlock" on page 24	Unlocks all resources.

How to use the functions is described in *"Administration"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestAllResourceUnlock

Call

```
b_errtype BestAllResourceUnlock( b_handletype handle );
```

Description Unlocks all locked resources (resets the internal counter incremented by each BestResourceLock call). This function can be used to re-establish the communication if the program hangs.

CLI Equivalent `BestAllResourceUnlock`

CLI Abbreviation `allresunlock`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestResourceUnlock" on page 29*
"BestResourceIsLocked" on page 27
"BestResourceLock" on page 28

BestCapabilityCheck

Call

```
b_errtype BestCapabilityCheck(
      b_handletype   handle,
      b_int32        capa_code );
```

Description Checks whether or not a capability of the testcard is enabled.

CLI Equivalent `BestCapabilityCheck capa_code=<capa_code>`

CLI Abbreviation `capachk code=<capa_code>`

Return Value **B_E_OK** – all checked capabilities are enabled.

B_E_NO_CAPABILITY – one or more capabilities are not enabled.

Input Parameters **handle** Handle to identify the session.

capa_code Value of OR-combined capability codes; see *"Capability Code Values" on page 25*.

See also *"BestCapabilityRead" on page 26*

Capability Code Values

Capability Code Value (CLI Abbreviation)	Check for
B_CAPABILITY_ALL (capaall, 0xFFFF F800)	all capabilities
B_CAPABILITY_NO (capano, 0)	no capability
B_CAPABILITY_64_BIT (capa64, 0x0010 0000\h)	64-bit capability
B_CAPABILITY_66_MHZ_AN (capa66an)	66-MHz-capability for the Analyzer
B_CAPABILITY_66_MHZ_EX (capa66ex)	66-MHz-capability for the Exerciser
B_CAPABILITY_ANALYZER (capaan, 0x0002 0000\h)	the Analyzer capability
B_CAPABILITY_EXERCISER (capaex, 0x0004 0000\h)	the Exerciser capability
B_CAPABILITY_HOSTINT (capahint, 0x0008 0000\h)	host interface capability (required for host access functions)
B_CAPABILITY_TRACEDEPTH_NONE (0x0000 0000\h) B_CAPABILITY_TRACEDEPTH_32k (0x0000 0100\h) B_CAPABILITY_TRACEDEPTH_64k (0x0000 0200\h) B_CAPABILITY_TRACEDEPTH_128k (0x0000 0300\h) B_CAPABILITY_TRACEDEPTH_256k (0x0000 0400\h) B_CAPABILITY_TRACEDEPTH_512k (0x0000 0500\h) B_CAPABILITY_TRACEDEPTH_1M (0x0000 0600\h) B_CAPABILITY_TRACEDEPTH_2M (0x0000 0700\h) B_CAPABILITY_TRACEDEPTH_4M (0x0000 0800\h)	trace memory capabilities

Capability Code Value (CLI Abbreviation)	Check for
B_CAPABILITY_PERFSEQ_NONE (0x0000 0000\h)	available performance measures
B_CAPABILITY_PERFSEQ_1 (0x0000 1000\h)	
B_CAPABILITY_PERFSEQ_2 (0x0000 2000\h)	
B_CAPABILITY_PERFSEQ_3 (0x0000 3000\h)	
B_CAPABILITY_PERFSEQ_4 (0x0000 4000\h)	
B_CAPABILITY_PERFSEQ_5 (0x0000 5000\h)	
B_CAPABILITY_PERFSEQ_6 (0x0000 6000\h)	
B_CAPABILITY_PERFSEQ_8 (0x0000 8000\h)	

BestCapabilityRead

Call

```
b_errtype BestCapabilityRead(
      b_handletype   handle,
      b_int32        *capa_code );
```

Description Reads the OR-combined capability codes.

CLI Equivalent `BestCapabilityRead`

CLI Abbreviation `caparead`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

Output Parameters **capa_code** Value of the OR-combined capability codes; see *"Capability Code Values" on page 25*.

See also *"BestCapabilityCheck" on page 24*

BestResourceIsLocked

Call

```
b_errtype BestResourceIsLocked(
      b_handletype      handle,
      b_resourcetype    resource,
      b_int32           *lock_count
      b_porttype        *lock_port );
```

Description Checks whether a resource has been locked by which port and how many times in reference to the current session.

CLI Equivalent `BestResourceIsLocked resource=<resource>`

CLI Abbreviation `resislocked res=<resource>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

resource Resource to be checked; see *"b_resourcetype" on page 263*.

Output Parameters

lock_count Number of calls that have already locked the resource. If the resource is unlocked, the value is 0.

lock_port Port that currently locks the resource; see *"b_porttype" on page 261*. If B_PORT_CURRENT is returned, the resource is locked by the current session.

See also

"BestResourceUnlock" on page 29
"BestAllResourceUnlock" on page 24
"BestResourceLock" on page 28
"BestPing" on page 22

BestResourceLock

Call

```
b_errtype BestResourceLock(
     b_handletype    handle,
     b_resourcetype  resource );
```

Description

Locks a resource.

This function fails if the resource is already locked by another port. The function does not fail if it is locked by the same port. Each time one resource is locked, an internal counter is incremented. This counter can be queried using *"BestResourceIsLocked" on page 27*.

CLI Equivalent

```
BestResourceLock resource=<resource>
```

CLI Abbreviation

```
reslock res=<resource>
```

Return Value

Error code; see *"b_errtype" on page 241*.

handle Handle to identify the session.

resource Resource to be locked; see *"b_resourcetype" on page 263*.

See also

"BestResourceUnlock" on page 29
"BestAllResourceUnlock" on page 24

BestResourceUnlock

Call

```
b_errtype BestResourceUnlock(
      b_handletype    handle,
      b_resourcetype  resource );
```

Description

Unlocks a resource gradually.

The internal counter is decremented. If the counter is at 0, the resource is unlocked.

CLI Equivalent

`BestResourceUnlock resource=<resource>`

CLI Abbreviation

`resunlock res=<resource>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

resource Resource to be unlocked; see *"b_resourcetype" on page 263*.

See also

"BestResourceIsLocked" on page 27
"BestResourceLock" on page 28
"BestAllResourceUnlock" on page 24

BestSystemInfoGet

Call

```
b_errtype BestSystemInfoGet(
      b_handletype       handle,
      b_systeminfotype  infoprop,
      b_int32           *value );
```

Description Reads information like bus width or bus speed on the PCI system under test.

CLI Equivalent `BestSystemInfoGet infoprop=<infoprop>`

CLI Abbreviation `siget prop=<infoprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

infoprop Information property to be read; see *"b_systeminfotype" on page 275*.

Output Parameters

value Value of the property; see *"b_systeminfotype" on page 275*.

See also *"BestVersionGet" on page 31*

BestVersionGet

Call

```
b_errtype BestVersionGet (
      b_handletype       handle,
      b_versionproptype  versionprop,
      b_charptrtype      *string );
```

Description Reads the version/date information stored on the EEPROMs of the testcards. This information is needed to check consistency between the firmware/HW and the C-API code.

CLI Equivalent `BestVersionGet versionprop=<versionprop>`

CLI Abbreviation `vget prop=<versionprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

versionprop Version property to be read; see *"b_versionproptype" on page 291*.

Output Parameters **string** Version or date string. This string is statically allocated and is overwritten each time this function is called.

See also *"BestSystemInfoGet" on page 30*

Power Up and Reset Control Functions

The following functions are used to control power up and reset:

Function	Result
"BestPowerUpPropSet" on page 38	Sets a power up property.
"BestPowerUpPropGet" on page 37	Gets a power up property.
"BestBoardPropStore" on page 36	Sets a hot swap power up property.
"BestBoardStoredPropGet" on page 37	Gets a hot swap power up property.
"BestAllPropStore" on page 34	Stores current settings as user defaults.
"BestAllPropLoad" on page 34	Loads user defaults as current settings.
"BestAllPropDefaultLoad" on page 33	Loads factory defaults as current settings.
"BestSMReset" on page 38	Resets all state machines.
"BestBoardReset" on page 36	Resets the testcard.
"BestBoardPropSet" on page 35	Sets a testcard property.
"BestBoardPropGet" on page 35	Gets a testcard property.

How to use the functions is described in *"Power-Up and Reset Control"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestAllPropDefaultLoad

Call `b_errtype BestAllPropDefaultLoad( b_handletype handle );`

Description Loads the factory defaults as current settings.

NOTE This function affects the current decoder settings and may not to be called while the card is under operation. First power off the PCI system under test or unplug the card.

Properties that affect settings of configuration space header registers will be altered only if the power up property B_PU_CONFRESTORE is set to 0. This property can be set with *"BestPowerUpPropSet" on page 38*.

CLI Equivalent `BestAllPropDefaultLoad`

CLI Abbreviation `aprpdefload`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestAllPropLoad" on page 34*
"BestPowerUpPropGet" on page 37
"b_puproptype" on page 262

BestAllPropLoad

Call `b_errtype BestAllPropLoad( b_handletype handle );`

Description Loads the user defaults as current settings.

NOTE Properties that affect settings of configuration space header registers will be altered only if the power up property B_PU_CONFRESTORE is set to 0. This property can be set with *"BestPowerUpPropSet" on page 38*.

CLI Equivalent `BestAllPropLoad`

CLI Abbreviation `aprpload`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestAllPropDefaultLoad" on page 33*
"BestPowerUpPropGet" on page 37
"b_puproptype" on page 262

BestAllPropStore

Call `b_errtype BestAllPropStore( b_handletype handle );`

Description Stores the current settings of the testcard as user defaults. The settings are then used after power up or after calling *"BestAllPropLoad" on page 34*.

CLI Equivalent `BestAllPropStore`

CLI Abbreviation `aprpstore`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestBoardPropStore" on page 36*

BestBoardPropGet

Call

```
b_errtype BestBoardPropGet(
      b_handletype     handle,
      b_boardproptype  boardprop,
      b_int32          *value );
```

Description Reads the testcard properties. They determine the testcard mode for PCI resets (RST# mode) and error reporting.

CLI Equivalent `BestBoardPropGet boardprop=<boardprop>`

CLI Abbreviation `bdprpget prop=<boardprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

boardprop Property to get; see *"b_boardproptype" on page 231*.

Output Parameters

value Value of the property; see *"b_boardproptype" on page 231*.

See also *"BestPowerUpPropGet" on page 37*

BestBoardPropSet

Call

```
b_errtype BestBoardPropSet(
      b_handletype     handle,
      b_boardproptype  boardprop,
      b_int32          value );
```

Description Sets the testcard properties. They determine the testcard mode for PCI resets (RST# mode) and error reporting.

CLI Equivalent `BestBoardPropSet boardprop=<boardprop> value=<value>`

CLI Abbreviation `bdprpset prop=<boardprop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

boardprop Property to be set; see *"b_boardproptype" on page 231*.

value Value to which the property is set; see *"b_boardproptype" on page 231*.

See also *"BestPowerUpPropSet" on page 38*

BestBoardPropStore

Call

```
b_errtype BestBoardPropStore(
      b_handletype     handle,
      b_boardproptype  boardprop,
      b_int32          value );
```

Description Stores the hot swap mode with which the card starts at the next power up.

CLI Equivalent `BestBoardPropStore boardprop=<boardprop> value=<value>`

CLI Abbreviation `bdprpstore prop=<boardprop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

boardprop Property to store; see *"b_boardproptype" on page 231*.

Output Parameters

value Value of the property; see *"b_boardproptype" on page 231*.

See also

"BestAllPropStore" on page 34
"BestBoardStoredPropGet" on page 37

BestBoardReset

Call

```
b_errtype BestBoardReset( b_handletype handle );
```

Description Performs a testcard reset. This is equivalent to re-powering the testcard. However, decoder and configuration space remain unchanged.

CLI Equivalent `BestBoardReset`

CLI Abbreviation `bdreset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

See also

"BestBoardReset" on page 36
"BestSMReset" on page 38

BestBoardStoredPropGet

Call

```
b_errtype BestBoardStoredPropGet(
      b_handletype     handle,
      b_boardproptype  boardprop,
      b_int32          *value );
```

Description This function reads the testcard's hot swap power up properties.

CLI Equivalent `BestBoardPropGet boardprop=<boardprop>`

CLI Abbreviation `bdprpget prop=<boardprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

boardprop Property to get; see *"b_boardproptype" on page 231*.

Output Parameters

value Value of the property; see *"b_boardproptype" on page 231*.

See also

"BestBoardPropStore" on page 36
"BestPowerUpPropGet" on page 37

BestPowerUpPropGet

Call

```
b_errtype BestPowerUpPropGet(
      b_handletype   handle,
      b_puproptype   pu_prop,
      b_int32        *value );
```

Description Reads the current value of a power up property.

CLI Equivalent `BestPowerUpPropGet pu_prop=<pu_prop> value=<value>`

CLI Abbreviation `puprpget prop=<pu_prop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

pu_prop Specifies the power up property to be read; see *"b_puproptype" on page 262*.

Output Parameters

value Value of the property; see *"b_puproptype" on page 262*.

See also

"BestPowerUpPropSet" on page 38
"BestBoardPropGet" on page 35

BestPowerUpPropSet

Call

```
b_errtype BestPowerUpPropSet(
      b_handletype   handle,
      b_puproptype   pu_prop,
      b_int32        value );
```

Description Sets a power up property.

CLI Equivalent `BestPowerUpPropSet pu_prop=<pu_prop> value=<value>`

CLI Abbreviation `puprpset prop=<pu_prop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

pu_prop Specifies the power up property to be set; see *"b_puproptype" on page 262*.

value Value to which the property is set; see *"b_puproptype" on page 262*.

See also

"BestPowerUpPropGet" on page 37
"BestBoardPropSet" on page 35

BestSMReset

Call `b_errtype BestSMReset( b_handletype handle );`

Description Resets the state machines of master, target, and analyzer. It ignores bus transactions that may currently be in progress. Therefore, it should be used only in the case of an error.

CLI Equivalent `BestSMReset`

CLI Abbreviation `smreset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

See also

"BestBoardReset" on page 36

Card Status Functions

The following functions are used to access the testcard's status register:

Function	Result
"BestStatusRegGet" on page 40	Reads the testcard's status register.
"BestStatusRegClear" on page 39	Clears bits in the testcard's status register.

How to use the functions is described in *"Card Status Register Access"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestStatusRegClear

Call

```
b_errtype BestStatusRegClear(
      b_handletype   handle,
      b_int32        clearpattern );
```

Description Clears bits in the testcard status register. The bits to be cleared are specified by a 1 in the bit mask (clear pattern).

CLI Equivalent `BestStatusRegClear clearpattern=<clearpattern>`

CLI Abbreviation `sregclear clear=<clearpattern>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

clearpattern Pattern specifying the bits to be cleared.

- 1 – clears the bit
- 0 – the bit remains unchanged

For example: `11111111\b` clears all bits.

See also *"BestStatusRegGet" on page 40*

BestStatusRegGet

Call

```
b_errtype BestStatusRegGet(
      b_handletype   handle,
      b_int32        *value );
```

Description Reads the testcard status register contents.

NOTE This is not the status register of the configuration space header.

CLI Equivalent `BestStatusRegGet`

CLI Abbreviation `sregget`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

Output Parameters **value** Contents of the status register; see *"Testcard Status Register" on page 41*.

See also *"BestStatusRegClear" on page 39*

Testcard Status Register

Type RO = read-only

Type RC = read/clear

Bit	Type	Default	Description	Value
[0]	RO	0	Master Run Bit	1= master active
[1]	RO	0	Target Active Bit	1= target active
[2]	RO	0	Observer Run Bit	Always 1
[3]	RO	0	Trace Run Bit	1= trace memory in run mode
[4]	RO	0	Protocol Error	1= protocol error detected
[5]	RC	0	Data Compare Error	1= data compare error detected
[6]	RC	0	Functional Error	1= functional error during command
[7]	RC	0	Block aborted.	1= at least one block has not been executed completely
[8]	RC	0	INTA asserted	1= asserted
[9]	RC	0	INTB asserted	1= asserted
[10]	RC	0	INTC asserted	1= asserted
[11]	RC	0	INTD asserted	1= asserted
[12]	RO	0	Test run failed	1= test run has failed
[13]	RO	0	Reserved	
[14]	RC	0	Self Traffic	1=master has accessed its own target
[15]	RO	0	Reserved	
[16]	RC	0	PCI Reset	1=PCI Reset has occurred
[17:31]	RO	0	Reserved	

PCI Analyzer Functions

The PCI Analyzer Functions are divided into the following sections:

Protocol Observer Functions

The following functions are used for the protocol observer:

Function	Result
"BestObsMaskSet" on page 49	Sets an individual error mask bit.
"BestObsMaskGet" on page 48	Reads an individual error mask bit.
"BestObsPropDefaultSet" on page 50	Sets all observer properties to their default values.
"BestObsStatusGet" on page 51	Reads the observer registers.
"BestObsErrResultGet" on page 46	Returns all errors found in the error registers in a text string.
"BestObsBitPositionFind" on page 45	Finds a set bit in the error registers.
"BestObsErrStringGet" on page 47	Returns the protocol rule text for a specific error.
"BestObsRuleErrTypeGet" on page 50	Returns the protocol rule identifier for a specific error.
"BestObsStatusClear" on page 51	Clears the current status of the observer.

How to use the functions is described in *"Protocol Observer Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestObsBitPositionFind

Call

```
b_errtype BestObsBitPositionFind(
      b_handletype   handle,
      b_int32        *errstat,
      b_int32        *errstat2,
      b_int32        *bitposition );
```

Description Returns the position of a bit set in the register pair with each call and resets this bit in the register pair. The register pair values can be requested with *"BestObsStatusGet" on page 51* and the bit position converted into a rule using *"BestObsRuleErrTypeGet" on page 50*.

CLI Equivalent No CLI equivalent.

CLI Abbreviation No CLI abbreviation.

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

errstat, errstat2 Error status register pair; see *"b_obsstatustype" on page 258*. This input parameters affect the output.

bitposition Position of the bit in the register pair. If no bit is set, the function returns 0xFFFFFFFF.

See also –

BestObsErrResultGet

Call

```
b_errtype BestObsErrResultGet(
      b_handletype   handle,
      b_int32        errstat,
      b_int32        errstat2,
      b_charptrtype  *errstring );
```

Description Returns a text string containing all errors found in the specified register pair. The register pair values can be requested with *“BestObsStatusGet” on page 51*.

CLI Equivalent `BestObsErrResultGet errstat=<errstat> errstat2=<errstat2>`

CLI Abbreviation `oeresultget errstat=<errstat> errstat2=<errstat2>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

errstat, errstat2 Error register pair; see *“b_obsstatustype” on page 258*.

Output Parameters

errstring Text string.

See also –

BestObsErrStringGet

Call

```
b_errtype BestObsErrStringGet(
      b_handletype    handle,
      b_int32         bitposition,
      b_charptrtype   *errstring );
```

Description

Returns the text string specified by the bit position of a rule violation, returned from *"BestObsStatusGet" on page 51*.

Example:

If a rule 2 violation is indicated by one of the error registers and you call this function with this value, the function will return "IRDY# must not be asserted on the same clock edge that FRAME# is asserted, but one or more clocks later".

CLI Equivalent

`BestObsErrStringGet bitposition=<bitposition>`

CLI Abbreviation

`oestrget pos=<bitposition>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

bitposition Bit position that specifies the rule.

Output Parameters

errstring Text string describing the rule.

See also

"b_obsruletype" on page 257

BestObsMaskGet

Call

```
b_errtype BestObsMaskGet(
      b_handletype   handle,
      b_obsruletype  obsrule,
      b_int32        *value );
```

Description Reads the mask bit of the specified rule.

CLI Equivalent `BestObsMaskGet obsrule=<obsrule>`

CLI Abbreviation `omget rule=<obsrule>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

obsrule Protocol rule identifier; see *“b_obsruletype” on page 257*.

Output Parameters

value Value of the mask bit:

- 0 (default) – rule not masked (enabled)
- 1 – rule masked (disabled)

See also *“BestObsMaskSet” on page 49*

BestObsMaskSet

Call

```
b_errtype BestObsMaskSet(
      b_handletype   handle,
      b_obsruletype  obsrule,
      b_int32        value );
```

Description Masks protocol rules, so that their violations are not indicated in the observer's first error register.

NOTE Violations of masked rules are not totally ignored: bit 2 of the observer status register is set and the rule violation is indicated in the accumulated error register. For status and error registers, see *"b_obsstatustype" on page 258*.

CLI Equivalent `BestObsMaskSet obsrule=<obsrule> value=<value>`

CLI Abbreviation `omset rule=<obsrule> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

obsrule Protocol rule to be masked; see *"b_obsruletype" on page 257*.

value Value to be entered in the mask:

- 0 (default) – rule not masked (enabled)
- 1 – rule masked (disabled)

See also *"BestObsMaskGet" on page 48*

BestObsPropDefaultSet

Call `b_errtype BestObsPropDefaultSet( b_handletype handle );`

Description Sets all mask bits to 0, so that all protocol rules will be observed, except SEM_8 and SEM_9, which are masked by default.

CLI Equivalent `BestObsPropDefaultSet`

CLI Abbreviation `oprpdefset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestObsMaskSet" on page 49*
"b_obsruletype" on page 257

BestObsRuleErrTypeGet

Call

```
b_errtype BestObsRuleErrTypeGet(
      b_handletype   handle
      b_int32        bitposition,
      b_obsruletype  *obsrule );
```

Description Returns the rule identifier specified by the given bit position.

Example:

If a rule 2 violation is indicated by one of the error registers and you call this function with this value, the function will return the numeric constant of B_R_IRDY_0. The numeric constants can be used as parameters in other function calls (for example in *"BestObsMaskGet" on page 48*).

CLI Equivalent `BestObsRuleErrTypeGet bitposition=<bitposition>`

CLI Abbreviation `osruleget pos=<bitposition>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

bitposition Bit position specifying the rule in one of the error registers.

Output Parameters **obsrule** Protocol rule identifier; see *"b_obsruletype" on page 257*.

See also –

BestObsStatusClear

Call `b_errtype BestObsStatusClear( b_handletype handle );`

Description Clears the observer status register, and the first and accumulated error registers; see *"b_obsstatustype" on page 258*.

CLI Equivalent `BestObsStatusClear`

CLI Abbreviation `osclear`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestObsStatusGet" on page 51*

BestObsStatusGet

Call

```
b_errtype BestObsStatusGet(
      b_handletype     handle,
      b_obsstatustype  obsstatus,
      b_int32          *value );
```

Description Reads the observer registers. Use this function to determine

- the first error that has occurred during a run,
- all errors that have occurred during a run, or
- the observer status (running/stopped, errors/no errors).

To convert the value passed back into a meaningful text string, use function *"BestObsErrStringGet" on page 47*.

To reset the registers, use the function *"BestObsStatusClear" on page 51*.

CLI Equivalent `BestObsStatusGet obsstatus=<obsstatus>`

CLI Abbreviation `osget stat=<obsstatus>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

obsstatus Observer register to be read; see *"b_obsstatustype" on page 258*.

Output Parameters **value** Value of the queried register.

See also –

Timing Check Functions

NOTE At present the timing check is only available for 33 MHz PCI busses.

The following functions are used to program the timing check:

Function	Result
"BestTimCheckGenPropSet" on page 55	Sets a generic timing check property.
"BestTimCheckGenPropGet" on page 54	Reads a generic timing check property.
"BestTimCheckMaskSet" on page 57	Sets a signal bit mask.
"BestTimCheckMaskGet" on page 56	Reads a signal bit mask.
"BestTimCheckDefaultSet" on page 53	Sets signal bit masks and timing check properties to default values.
"BestTimCheckStatusGet" on page 63	Reads the current status of the timing check.
"BestTimCheckStatusClear" on page 62	Clears the status registers of the timing check.
"BestTimCheckPropSet" on page 60	Sets setup time and hold time values in the preparation register.
"BestTimCheckPropGet" on page 59	Reads setup time and hold time values from the preparation register.
"BestTimCheckRead" on page 61	Copies the timing check properties from the card to the preparation register.
"BestTimCheckProg" on page 58	Copies the timing check properties from the preparation register to the card.
"BestTimCheckResultGet" on page 61	Returns the current results of the timing check in a text string.

How to use the functions is described in *"Timing Check Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide.*

BestTimCheckDefaultSet

Call `b_errtype BestTimCheckDefaultSet( b_handletype handle );`

Description Sets the following properties to default values:

- signal bit mask
- timing check properties
- generic timing check properties

CLI Equivalent `BestTimCheckDefaultSet`

CLI Abbreviation `tcdefset`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

See also
“BestTimCheckMaskSet” on page 57
“BestTimCheckPropSet” on page 60
“BestTimCheckGenPropSet” on page 55
“b_tcproptype” on page 282
“b_tcgenproptype” on page 281

BestTimCheckGenPropGet

Call

```
b_errtype BestTimCheckGenPropGet(
      b_handletype      handle,
      b_tcgenproptype   tcgenprop,
      b_int32           *value );
```

Description

Reads a generic timing check property.

The generic timing check property determines whether the PCI Specification values of setup and hold time or the values in the preparation register are used.

CLI Equivalent

```
BestTimCheckGenPropGet tcgenprop=<tcgenprop>
```

CLI Abbreviation

```
tcgprpget prop=<tcgenprop>
```

Return Value

Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

tcgenprop Generic property to be read; see *“b_tcgenproptype” on page 281*.

Output Parameters

value Value of the generic property; see *“b_tcgenproptype” on page 281*.

See also

“BestTimCheckDefaultSet” on page 53
“BestTimCheckGenPropSet” on page 55

BestTimCheckGenPropSet

Call

```
b_errtype BestTimCheckGenPropSet(
      b_handletype      handle,
      b_tcgenproptype   tcgenprop,
      b_int32           value );
```

Description Sets a generic timing check property.

The generic timing check property determines whether the PCI Specification values of setup and hold time or the values in the preparation register are used.

CLI Equivalent `BestTimCheckGenPropSet tcgenprop=<tcgenprop> value=<value>`

CLI Abbreviation `tcgprpset prop=<tcgenprop> val=<value>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

tcgenprop Generic property to be set; see *“b_tcgenproptype” on page 281*.

value Value of the generic property; see *“b_tcgenproptype” on page 281*.

See also *“BestTimCheckDefaultSet” on page 53*
“BestTimCheckGenPropGet” on page 54

BestTimCheckMaskGet

Call

```
b_errtype BestTimCheckMaskGet(
      b_handletype   handle,
      b_signaltype   signal,
      b_int32        *value );
```

Description Reads the bit mask of a signal.

CLI Equivalent `BestTimCheckMaskGet signal=<signal>`

CLI Abbreviation `tcmget sig=<signal>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

signal Signal of which the bit mask is to be read; see *"b_signaltype (for Timing Check)" on page 264*.

Output Parameters

value Bit mask of each signal. The signal lengths can range from 1 to 32 bits (for example, signal AD32). The value contains valid mask bits matching the length of the signal.

Value of the mask bit:

- 0 (default) – not masked (enabled)
- 1 – masked (disabled)

See also *"BestTimCheckDefaultSet" on page 53*
"BestTimCheckMaskSet" on page 57

BestTimCheckMaskSet

Call

```
b_errtype BestTimCheckMaskSet(
      b_handletype   handle,
      b_signaltype   signal,
      b_int32        value );
```

Description

Sets the bit mask for a signal.

A masked signal can no longer be used as a trigger, but the violation will be shown in the timing check report.

CLI Equivalent

`BestTimCheckMaskSet signal=<signal> value=<value>`

CLI Abbreviation

`tcmset sig=<signal> val=<value>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

signal Signal to be masked; see *"b_signaltype (for Timing Check)" on page 264*.

value Bit mask of each signal. The signal lengths can range from 1 to 32 bits (for example, signal AD32). The value must contains valid mask bits matching the length of the signal.

Value of the mask bit:

- 0 (default) – not masked (enabled)
- 1 – masked (disabled)

See also

"BestTimCheckDefaultSet" on page 53
"BestTimCheckMaskGet" on page 56

BestTimCheckProg

Call `b_errtype BestTimCheckProg( b_handletype handle );`

Description Writes the timing check properties from the preparation register to the testcard. This function also clears the timing check status register and performs a consistency check on the preparation register contents.

Before calling this function, use *"BestTimCheckPropSet" on page 60* to program the preparation register.

NOTE If you want to program your own setup time and hold time values from the preparation register instead of the time values of the PCI Specification, you first need to set the generic timing check property B_TCGEN_SPEC to 0. To set this property, use *"BestTimCheckGenPropSet" on page 55*.

CLI Equivalent `BestTimCheckProg`

CLI Abbreviation `tcprog`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also
"b_tcgenproptype" on page 281
"b_tcproptype" on page 282
"b_tcstatustype" on page 282
"BestTimCheckRead" on page 61

BestTimCheckPropGet

Call

```
b_errtype BestTimCheckPropGet(
      b_handletype   handle,
      b_tcproptype   tcprop,
      b_int32        *value );
```

Description Reads the setup time and the hold time values from the preparation register.

To get current values, first use *"BestTimCheckRead" on page 61* to read the timing check properties from the testcard to the preparation register.

CLI Equivalent `BestTimCheckPropGet tcprop=<tcprop>`

CLI Abbreviation `tcprpget prop=<tcprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

tcprop Property to be read; see *"b_tcproptype" on page 282*.

Output Parameters

value Value of the property; see *"b_tcproptype" on page 282*.

See also *"BestTimCheckDefaultSet" on page 53*
"BestTimCheckPropSet" on page 60

BestTimCheckPropSet

Call

```
b_errtype BestTimCheckPropSet(
      b_handletype   handle,
      b_tcproptype   tcprop,
      b_int32        value );
```

Description Defines the setup time and the hold time values in the preparation register.

To write these values to the testcard, use *"BestTimCheckProg" on page 58*.

NOTE Setting these values takes effect only if the property B_TCGEN_SPEC is set to 0. To set this property, use *"BestTimCheckGenPropSet" on page 55*.

CLI Equivalent `BestTimCheckPropSet tcprop=<tcprop> value=<value>`

CLI Abbreviation `tcprpset prop=<tcprop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

tcprop Property to be set; see *"b_tcproptype" on page 282* .

value Value of the property; see *"b_tcproptype" on page 282*.

See also *"BestTimCheckDefaultSet" on page 53*
"BestTimCheckPropGet" on page 59

BestTimCheckRead

Call
```
b_errtype BestTimCheckRead( b_handletype handle );
```

Description Reads the timing check properties from the testcard to the preparation register.

To read the timing check properties from the preparation register, use *"BestTimCheckPropGet" on page 59*.

CLI Equivalent `BestTimCheckRead`

CLI Abbreviation `tcread`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also
"b_tcproptype" on page 282
"BestTimCheckProg" on page 58

BestTimCheckResultGet

Call
```
b_errtype BestTimCheckResultGet(
      b_handletype      handle,
      b_charptrtype     *errortext );
```

Description Returns a text string containing the violated signals.

NOTE Before using this function, ensure that the timing check can be performed properly under the current conditions. Use *"BestTimCheckStatusGet" on page 63* to query B_TC_ERROR of property B_TC_TCSTAT. If the error flag is set, the returned text string may contain wrong information.

CLI Equivalent `BestTimCheckResultGet`

CLI Abbreviation `tcresultget`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

Output Parameters **errortext** Text string.

See also *"b_tcstatustype" on page 282*

BestTimCheckStatusClear

Call `b_errtype BestTimCheckStatusClear( b_handletype handle );`

Description Clears the status register contents (timing check result, violation flag) of the timing check.

NOTE The flag indicating instable frequency/calibration error B_TC_ERROR remains uncleared. To clear this bit, first eliminate the error. Then execute *"BestTimCheckProg" on page 58*. The flag will be cleared, if the function has been executed successfully.

CLI Equivalent `BestTimCheckStatusClear`

CLI Abbreviation `tcsclear`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestTimCheckStatusGet" on page 63*
"b_tcstatustype" on page 282

BestTimCheckStatusGet

Call

```
b_errtype BestTimCheckStatusGet(
      b_handletype      handle,
      b_tcstatustype    status,
      b_int32           *value );
```

Description Reads the status register of the timing check.

Use this function to determine whether the timing check can be performed properly under the current conditions and whether signals have been violated.

The timing check cannot be performed properly if

- the card is not calibrated,
- the bus frequency has changed since the last successful execution of *"BestTimCheckProg" on page 58*,
- the bus frequency is instable.

NOTE Always use this function before getting timing check results with *"BestTimCheckResultGet" on page 61*.

CLI Equivalent `BestTimCheckStatusGet tcstatus=<status>`

CLI Abbreviation `tcsget stat=<status>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

status Timing check register to be read; see *"b_tcstatustype" on page 282*.

Output Parameters **value** Value of the register; see *"b_tcstatustype" on page 282*.

See also –

Pattern Term Function

The following function is used for the pattern terms:

Function	Result
"BestPattSet" on page 64	Programs a pattern term.

How to use the functions is described in *"Programming the Pattern Terms"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestPattSet

Call

```
b_errtype BestPattSet(
      b_handletype   handle,
      b_int32        pattern_ident,
      b_charptrtype  pattern );
```

Description Specifies a pattern term. This term can then be used in the condition strings of a sequencer description table.

CLI Equivalent `BestPattSet pattern_ident=<pattern_ident> pattern=<pattern>`

CLI Abbreviation `pattset pid=<pattern_ident> patt=<pattern>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

pattern_ident Pattern term identifier; see *"Pattern Term Identifiers" on page 65*.

pattern String containing the logical expression for the condition. The string must be set in quotation marks.

To build pattern terms see

- *"Syntax Diagrams for Numbers and 10XNumbers" on page 66*
- *"Standard Pattern Term Operators" on page 66* and *"Standard Pattern Term Syntax Diagram" on page 67*
- *"Transitional Pattern Term Operators" on page 68* and *"Transitional Pattern Term Syntax Diagram" on page 68*

For specifiable signals, see *"b_signaltype (List of Signals)" on page 265*.

See also –

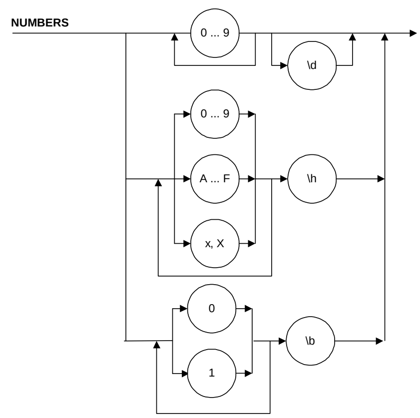
Pattern Term Identifiers

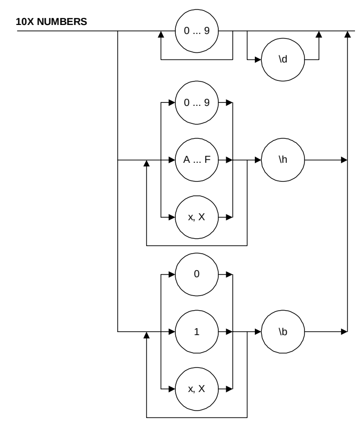
These identifiers are used to select the pattern term to be programmed when using the function *"BestPattSet" on page 64*.

Identifier (CLI Abbreviation)	Description
B_PATT_TERM_0 (pt0)	Both transitional and standard pattern term (to specify this, use *"BestTracePropSet" on page 83*). Used for trace memory triggering only.
B_PATT_TERM_1 (pt1) ... B_PATT_TERM_23 (pt23)	Standard pattern terms. Used for trace memory triggering, performance analysis, and master conditional start.

Syntax Diagrams for Numbers and 10XNumbers

To program pattern terms, hexadecimal, decimal and binary numbers can be used. Signals of type “10X” additionally allow the use of “Don’t Cares” (= x or X). This is expressed in the following syntax diagrams:





These numbers can be used when building logical expressions to program a standard or transitional pattern term.

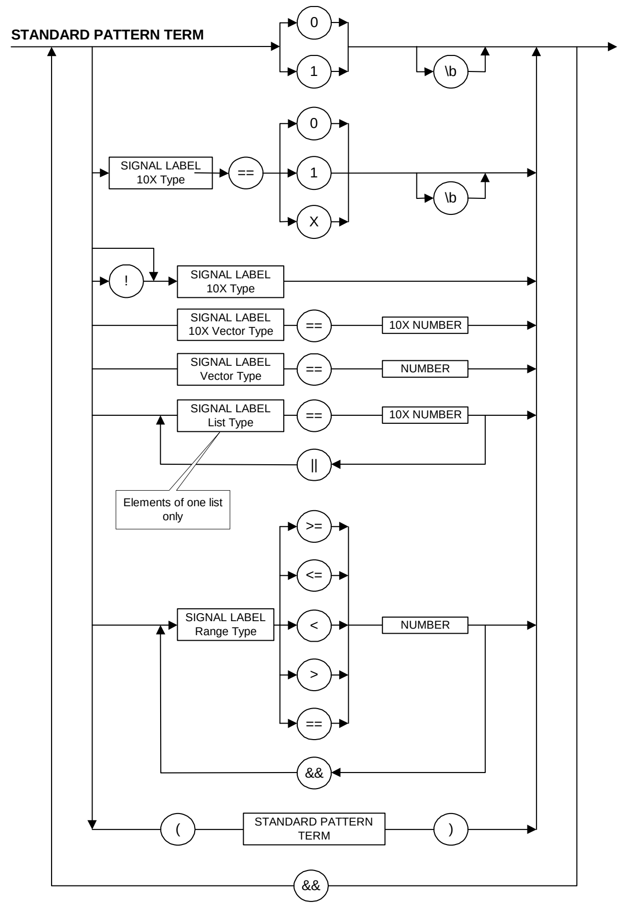
Standard Pattern Term Operators

The following table shows the operators that can be used in standard pattern terms to combine the signal labels. The operators appear in the order of their priority:

Operator	Operation	Applicable
!	negation	for all signals
==	compare for equality	for all signals
&&	logical AND	for all signals

Signals of “10X vector type” can be queried bitwise; see the examples in *“Standard Pattern Term Syntax Diagram” on page 67*.

Standard Pattern Term Syntax Diagram



Examples

- "b_state==3\h && AD32==b8xxx\h"

  Detects address phases and addresses in video memory range.

- "FRAME && (b_state==2 || b_state==3) && (AD32==b8xxx\h)"

Transitional Pattern Term Operators

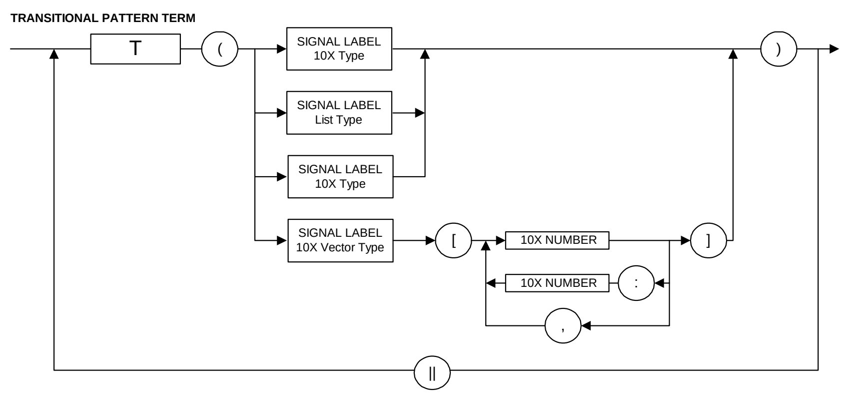
To build the transitional pattern term, use the transition function `T()` to determine whether a signal changes its state. Only `pt0` can be used as transitional pattern term.

Any signal from the “Signal” column of the table in *b_signaltype (List of Signals)* may be specified as an argument for this function. Signals of “10X vector type” can be queried bitwise. See the examples in *“Transitional Pattern Term Syntax Diagram” on page 68*.

If multiple signals are to be queried, they can be combined via logical OR:

Operator	Operation	Applicable
||	logical OR	for all signals

Transitional Pattern Term Syntax Diagram



Examples:

- `"T(GNT) || T(REQ)"`

  Detects transitions of the GNT# or REQ# lines.

- `"T(AD32[31:17, 4:2])"`

  Detects toggling signals on address/data lines 2 to 4 and 17 to 31.

Trace Memory Trigger Sequencer Functions

The following functions are used to program the trace memory trigger sequencer:

Function	Result
"BestTrigSeqGenPropDefaultSet" on page 70	Sets the preload value of the feedback counter to the default value.
"BestTrigSeqGenPropGet" on page 70	Reads the preload value of the feedback counter.
"BestTrigSeqGenPropSet" on page 71	Sets the preload value of the feedback counter.
"BestTrigSeqPropDefaultSet" on page 72	Sets all properties in the trigger sequencer description table to default values.
"BestTrigSeqTranPropDefaultSet" on page 75	Sets all properties of a transient in the trigger sequencer description table to default values.
"BestTrigSeqTranPropSet" on page 76	Sets a numeric transition property ("state" or "next state").
"BestTrigSeqTranCondPropSet" on page 73	Sets a condition in the trigger sequencer description table.
"BestTrigSeqProg" on page 71	Writes the sequencer description table to sequencer memory.

How to use the functions is described in *"Sequencer Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestTrigSeqGenPropDefaultSet

Call

```
b_errtype BestTrigSeqGenPropDefaultSet( b_handletype handle );
```

Description Sets the value of the generic trigger sequencer property to 0. This property determines the preload value of a trace memory trigger sequencer feedback counter.

CLI Equivalent `BestTrigSeqGenPropDefaultSet`

CLI Abbreviation `tsgenprpdefset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestTrigSeqGenPropGet" on page 70*
"BestTrigSeqGenPropSet" on page 71
"b_trigseqgenproptype" on page 289

BestTrigSeqGenPropGet

Call

```
b_errtype BestTrigSeqGenPropGet(
      b_handletype          handle,
      b_trigseqgenproptype  trigseqgenprop,
      b_int32               *value );
```

Description Reads the value of the generic trigger sequencer property. This property determines the preload value of the trace memory trigger sequencer feedback counter.

CLI Equivalent `BestTrigSeqGenPropGet trigseqgenprop=<trigseqgenprop>`

CLI Abbreviation `tsgenprpget prop=<trigseqgenprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

trigseqgenprop Property to be read; see *"b_trigseqgenproptype" on page 289*.

Output Parameters **value** Value of the property; see *"b_trigseqgenproptype" on page 289*.

See also *"BestTrigSeqGenPropDefaultSet" on page 70*
"BestTrigSeqGenPropSet" on page 71

BestTrigSeqGenPropSet

Call

```
b_errtype BestTrigSeqGenPropSet(
      b_handletype          handle,
      b_trigseqgenproptype  trigseqgenprop,
      b_int32               value );
```

Description Sets the value of the generic trigger sequencer property. This property determines the preload value of the trace memory trigger sequencer feedback counter.

CLI Equivalent `BestTrigSeqGenPropSet trigseqgenprop=<trigseqgenprop> value=<value>`

CLI Abbreviation `tsgenprpset prop=<trigseqgenprop> val=<value>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

trigseqgenprop Property to be set; see *“b_trigseqgenproptype” on page 289*.

value Value the property is set to; see *“b_trigseqgenproptype” on page 289*.

See also

“BestTrigSeqGenPropDefaultSet” on page 70
“BestTrigSeqGenPropGet” on page 70

BestTrigSeqProg

Call `b_errtype BestTrigSeqProg( b_handletype handle );`

Description Writes the information stored in the trigger sequencer description table to the sequencer memory.

This function also checks whether the transition conditions of one state are consistent. If they are not, the function returns an error.

CLI Equivalent `BestTrigSeqProg`

CLI Abbreviation `tsprog`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

See also –

BestTrigSeqPropDefaultSet

Call `b_errtype BestTrigSeqPropDefaultSet( b_handletype handle );`

Description Initializes the trace memory trigger sequencer description table and sets all properties to default values.

CLI Equivalent `BestTrigSeqPropDefaultSet`

CLI Abbreviation `tsprpdefset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle	Handle to identify the session.

See also *"BestTrigSeqTranPropDefaultSet" on page 75*

BestTrigSeqTranCondPropSet

Call

```
b_errtype BestTrigSeqTranCondPropSet(
      b_handletype              handle,
      b_int32                   transient,
      b_trigseqtrancondproptype trigseqtrancondprop,
      b_charptrtype             condition );
```

Description Sets a condition in the trace memory trigger sequencer description table.

The condition property can be the transition condition, trigger condition, storage qualifier condition, or conditions to decrement and preload the feedback counter.

CLI Equivalent

```
BestTrigSeqTranCondPropSet transient=<transient>
trigseqtrancondprop=<trigseqtrancondprop> condition=<condition>
```

CLI Abbreviation

```
tstrancprpset tran=<transient> prop=<trigseqtrancondprop>
con=<condition>
```

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

transient Transient number (0 … 255).

trigseqtrancondprop Condition property to be set; see *“b_trigseqtrancondproptype” on page 290*.

condition Condition string. The string must be set in quotation marks. See *“Conditions Reference” on page 74*.

See also *“BestTrigSeqTranPropDefaultSet” on page 75*
“BestTrigSeqTranPropSet” on page 76

Conditions Reference

Conditions are used in the sequencer description tables.

A condition is either true (1) or false (0) and controls the behavior of the specified sequencer. Conditions are specified by means of a logical expression (condition string).

These condition strings can consist of:

- Pattern term identifiers (pt0 ... pt23); see *"Pattern Term Identifiers" on page 65*.
- The terminal count of the sequencer feedback counter (tc).
- Logical operators (logical AND, logical OR, …).
- True or false settings ("1" or "0").

Example:

- `"( !pt0 || pt1 || pt2 ) && tc"`

  Valid condition string to program the trigger sequencer.

Pattern Identifiers for Sequencers

The following table shows the pattern identifiers that can be used in condition strings for the different sequencers:

Identifier	Description
pt0	Used as a transitional or standard pattern term (see *"b_traceproptype" on page 285*).
pt1 … pt23	Used as standard pattern terms only.
tc	Terminal count of its feedback counter.
tcc	Terminal count of trigger IO feedback counter C
tcd	Terminal count of trigger IO feedback counter D

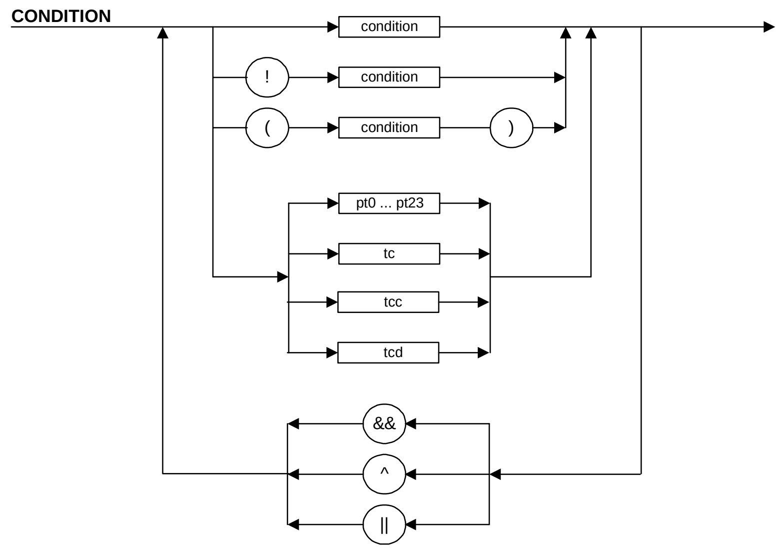
Logical Operators

The following table shows the logical operators in the order of their priority:

Operator	Operation
!	negation
&&	logical AND
^	logical XOR
||	logical OR

"(" and ")" can be used to override priorities.

Condition Syntax Diagram The syntax of the condition strings is expressed in the following diagram:



BestTrigSeqTranPropDefaultSet

Call

```
b_errtype BestTrigSeqTranPropDefaultSet(
      b_handletype   handle,
      b_int32        transient );
```

Description Sets all properties of a transient in the trace memory trigger sequencer description table to default values.

For a description of properties and default values, refer to *"b_trigseqtranproptype" on page 290* and *"b_trigseqtrancondproptype" on page 290*.

CLI Equivalent `BestTrigSeqTranPropDefaultSet transient=<transient>`

CLI Abbreviation `tstranprpdefset tran=<transient>`

Return Value Error code; see *"b_errtype" on page 241*

Input Parameters

handle Handle to identify the session.

transient Transient number (0 … 255).

See also *"BestTrigSeqTranPropSet" on page 76*
"BestTrigSeqTranCondPropSet" on page 73

BestTrigSeqTranPropSet

Call

```
b_errtype BestTrigSeqTranPropSet(
      b_handletype           handle,
      b_int32                transient,
      b_trigseqtranproptype  trigseqtranprop,
      b_int32                value );
```

Description Sets a numeric transition property (“state” or “next state”) in the trace memory trigger sequencer description table.

CLI Equivalent `BestTrigSeqTranPropSet transient=<transient> trigseqtranprop=<trigseqtranprop> value=<value>`

CLI Abbreviation `tstranprpset tran=<transient> prop=<trigseqtranprop> val=<value>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

transient Transient number (0 … 255).

trigseqtranprop Property to be set; see *“b_trigseqtranproptype” on page 290*.

value Value the property is set to, *“b_trigseqtranproptype” on page 290*.

See also

“BestTrigSeqTranPropDefaultSet” on page 75
“BestTrigSeqTranCondPropSet” on page 73

Trace Memory Functions

The following functions are used for the trace memory:

Function	Result
"BestAnalyzerRun" on page 77	Starts the PCI analyzer trace memory.
"BestAnalyzerStop" on page 78	Stops the PCI analyzer trace memory.
"BestTracePropSet" on page 83	Sets a property for the trace memory (for compatibility reasons).
"BestTraceRun" on page 84	Enables trace memory.
"BestTraceStop" on page 85	Stops the current trace run.
"BestTraceDataGet" on page 81	Loads trace memory lines from the testcard.
"BestTraceStatusGet" on page 84	Reads the trace status register, the line number of the trigger event, and the number of lines captured.
"BestTraceBitPosGet" on page 79	Returns the position and length of a signal in a trace memory line.
"BestTraceBytePerLineGet" on page 80	Returns the number of bytes of a trace memory line.
"BestTracePattPropSet" on page 82	Sets compare patterns for trace memory control.

How to use the functions is described in *"Trace Memory Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestAnalyzerRun

Call

```
b_errtype BestAnalyzerRun( b_handletype handle );
```

Description

Starts the PCI analyzer trace memory. This function is equivalent to *"BestTraceRun" on page 84*.

CLI Equivalent

```
BestAnalyzerRun
```

CLI Abbreviation

```
arun
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

See also

"BestAnalyzerStop" on page 78

BestAnalyzerStop

Call `b_errtype BestAnalyzerStop( b_handletype handle );`

Description Stops the PCI analyzer (protocol observer and trace memory). The current run status information and trace memory content are not affected.

The trace memory contains 100% pre-trigger history and is ready to be uploaded.

This function is equivalent to *"BestTraceStop" on page 85*.

CLI Equivalent `BestAnalyzerStop`

CLI Abbreviation `astop`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestAnalyzerRun" on page 77*

BestTraceBitPosGet

Call

```
b_errtype BestTraceBitPosGet(
      b_handletype   handle,
      b_signaltype   signal,
      b_int32        *position,
      b_int32        *length );
```

Description Returns the position and length of the specified signal in trace memory lines.

CLI Equivalent `BestTraceBitPosGet signal=<signal>`

CLI Abbreviation `trcbtposget signal=<signal>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

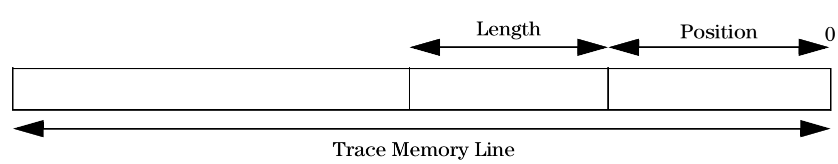
handle Handle to identify the session.

signal Signal of which the position and length are to be determined; see *"b_signaltype (List of Signals)" on page 265*.

Output Parameters

position Bit position of the specified signal within a trace memory line. This value is the offset (in bits) from the least significant bit to the first bit of the specified signal.

length Length in bits of the signal data. This is the width of the signal.



See also *"BestTraceBytePerLineGet" on page 80*

BestTraceBytePerLineGet

Call

```
b_errtype BestTraceBytePerLineGet(
      b_handletype   handle,
      b_int32        *bytes_per_line );
```

Description Returns the number of bytes in a trace memory line. This value depends on the Agilent E2920 hardware actually used.

CLI Equivalent `BestTraceBytePerLineGet`

CLI Abbreviation `trcbtplget`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

Output Parameters **bytes_per_line** Number of bytes per line of trace data.

See also *“BestTraceBitPosGet” on page 79*

BestTraceDataGet

Call

```
b_errtype BestTraceDataGet(
      b_handletype   handle,
      b_int32        startline,
      b_int32        n_of_lines,
      b_int32        *data );
```

Description Loads trace memory lines from the testcard.

CLI Equivalent `BestTraceDataGet startline=<startline> n_of_lines=<n_of_lines>`

CLI Abbreviation `trcdget start=<startline> n_of_lines=<n_of_lines>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

startline First line to be loaded from trace memory.

n_of_lines Number of lines to be loaded from trace memory.

Output Parameters

data Array of 32-bit values (dwords) containing the data read from trace memory.

The required size of this array can be calculated in the following way:

- Size of the array in bytes =

  number of lines (n_of_lines) × number of bytes per line;

  (the number of bytes per line can be requested with *"BestTraceBytePerLineGet" on page 80*)

- Size of the array in dwords =

  size of the array in bytes / 4

See also –

BestTracePattPropSet

Call

```
b_errtype BestTracePattPropSet(
      b_handletype         handle,
      b_tracepattproptype  tracepattprop,
      b_charptrtype        pattern );
```

Description Sets compare patterns for trace memory control.

CLI Equivalent

```
BestTracePattPropSet tracepattprop=<tracepattprop>
pattern=<pattern>
```

CLI Abbreviation

```
trcpprpset prop=<tracepattprop> patt=<pattern>
```

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

tracepattprop Property to be set; see *"b_tracepattproptype" on page 284*.

pattern Logical expression for the condition; see *"b_tracepattproptype" on page 284*. The string must be set in quotation marks.

See also –

BestTracePropSet

Call

```
b_errtype BestTracePropSet(
      b_handletype      handle,
      b_traceproptype   traceprop,
      b_int32           value );
```

Description Sets a property for the trace memory.

CLI Equivalent `BestTracePropSet traceprop=<traceprop> value=<value>`

CLI Abbreviation `trcprpset prop=<traceprop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

traceprop Property to be set; see *"b_traceproptype" on page 285*.

value Value the property is set to; see *"b_traceproptype" on page 285*.

See also –

BestTraceRun

Call

```
b_errtype BestTraceRun( b_handletype handle );
```

Description

Enables the trace memory. Data is acquired according to the trace memory trigger sequencer. This function is equivalent to *"BestAnalyzerRun" on page 77*.

CLI Equivalent

`BestTraceRun`

CLI Abbreviation

`trcrun`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

See also

"BestTraceStop" on page 85

BestTraceStatusGet

Call

```
b_errtype BestTraceStatusGet(
      b_handletype        handle,
      b_tracestatustype   tracestatus
      b_int32             *status );
```

Description

Reads the following data from the trace memory:

- trace status register,
- line number of the trigger event, or
- number of lines captured.

It is intended to be used to control the trace run.

CLI Equivalent

`BestTraceStatusGet tracestatus=<tracestatus>`

CLI Abbreviation

`tsget stat=<tracestatus>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

tracestatus Status to be read; see *"b_tracestatustype" on page 286*.

Output Parameters

status Value of the queried status; see *"b_tracestatustype" on page 286*.

See also

–

BestTraceStop

Call

```
b_errtype BestTraceStop( b_handletype handle );
```

Description

Stops the current trace run. The current run status information is not affected.

The trace memory contains 100% pre-trigger history and is ready to be uploaded.

This function is equivalent to *“BestAnalyzerStop” on page 78*.

CLI Equivalent

`BestTraceStop`

CLI Abbreviation

`trcstop`

Return Value

Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

See also

“BestTraceRun” on page 84

Performance Measure Functions

Setup and Programming

The functions of this category are used to set up and program the performance measures and the corresponding sequencers.

Function	Result
"BestPerfGenPropDefaultSet" on page 88	Sets the generic performance properties to default values.
"BestPerfGenPropSet" on page 90	Sets the value of a generic performance measure property.
"BestPerfGenPropGet" on page 89	Reads the value of a generic performance measure property.
"BestPerfSeqPropDefaultSet" on page 92	Initializes a sequencer description table of a performance measure.
"BestPerfSeqTranPropDefaultSet" on page 94	Sets all properties of a transient to default values.
"BestPerfSeqTranPropSet" on page 95	Sets a numeric transition property ("state" or "next state").
"BestPerfSeqTranCondPropSet" on page 93	Sets a condition in the sequencer description table of a performance measure.
"BestPerfSeqProg" on page 91	Writes the sequencer description table of a performance measure to sequencer memory.

Running and Evaluating

The following functions are used to run and evaluate the measurement.

Function	Result
"BestPerfRun" on page 91	Starts all counters.
"BestPerfStatusGet" on page 96	Reads the status of the selected measure.
"BestPerfStop" on page 97	Stops all counters.
"BestPerfUpdate" on page 97	Captures the values of all performance counters.
"BestPerfCtrRead" on page 87	Reads counter values.

How to use the functions is described in *"Performance Measurement Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide.*

BestPerfCtrRead

Call

```
b_errtype BestPerfCtrRead(
      b_handletype   handle,
      b_int32        measure,
      b_int32        counter_id,
      b_int32        *value );
```

Description Reads the counter values of a performance measure.

The counter values must have been previously updated with *"BestPerfUpdate" on page 97*.

CLI Equivalent `BestPerfCtrRead measure=<measure> counter_id=<counter_id>`

CLI Abbreviation `pctrread meas=<measure> cid=<counter_id>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

measure Value to identify the measure; see *"Measure Identifier" on page 87*.

counter_id Value to identify the counter; see *"Counter Identifier" on page 88*.

Output Parameters

value Value of the queried counter.

See also –

Measure Identifier

Measure Identifier (CLI Abbreviation)	Description
B_PERFMEAS_0 (meas0, 0) … B_PERFMEAS_7 (meas7, 7)	Selects one of the performance measures 0 … 7.

Counter Identifier

Counter Identifier (CLI Abbreviation)	Description
B_PERFCTR_A (ctra) B_PERFCTR_A_HI (ctrahi)	Nominator counter value (lower and upper 32 bits).
B_PERFCTR_B (ctrb) B_PERFCTR_B_HI (ctrbhi)	Denominator counter value (lower and upper 32 bits).
B_PERFCTR_C (ctrc) B_PERFCTR_C_HI (ctrchi)	Feedback counter C value (lower and upper 32 bits).
B_REFCTR (refc) B_REFCTR_HI (refchi)	PCI clock reference counter value (lower and upper 32 bits).

BestPerfGenPropDefaultSet

Call

```
b_errtype BestPerfGenPropDefaultSet(
      b_handletype   handle,
      b_int32        measure );
```

Description

Sets the generic performance properties of a measure to default values. These properties are used to

- determine the mode used to increment the counter A,
- set the preload value of the feedback counter.

For generic properties and default values, refer to *"b_perfgenproptype" on page 259*.

CLI Equivalent

```
BestPerfGenPropDefaultSet measure=<measure>
```

CLI Abbreviation

```
pgenprpdefset meas=<measure>
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

measure Value to identify the measure; see *"Measure Identifier" on page 87*.

See also

"BestPerfGenPropGet" on page 89
"BestPerfGenPropSet" on page 90

BestPerfGenPropGet

Call

```
b_errtype BestPerfGenPropGet(
      b_handletype       handle,
      b_int32            measure,
      b_perfgenproptype  perfgenprop,
      b_int32            *value );
```

Description

Reads the value of a generic performance property of a measure. These properties are used to

- determine the mode used to increment the nominator counter,
- set the preload value of the feedback counter.

CLI Equivalent

`BestPerfGenPropGet measure=<measure> perfgenprop=<perfgenprop>`

CLI Abbreviation

`pgenprpget meas=<measure> prop=<perfgenprop>`

Return Value

Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

measure Value to identify the measure; see *“Measure Identifier” on page 87*.

perfgenprop Property to be queried; see *“b_perfgenproptype” on page 259*.

Output Parameters

value Value of the queried property; see *“b_perfgenproptype” on page 259*.

See also

“BestPerfGenPropDefaultSet” on page 88
“BestPerfGenPropSet” on page 90

BestPerfGenPropSet

Call

```
b_errtype BestPerfGenPropSet(
      b_handletype       handle,
      b_int32            measure,
      b_perfgenproptype  perfgenprop,
      b_int_32            value );
```

Description

Sets the value of a generic performance property of a measure. These properties are used to

- determine the mode used to increment the nominator counter,
- set the preload value of the feedback counter.

CLI Equivalent

`BestPerfGenPropSet measure=<measure> perfgenprop=<perfgenprop> value=<value>`

CLI Abbreviation

`pgenprpset meas=<measure> prop=<perfgenprop> val=<value>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

measure Value to identify the measure; see *"Measure Identifier" on page 87*.

perfgenprop Property to be set; see *"b_perfgenproptype" on page 259*.

value Value to which the property is set; see *"b_perfgenproptype" on page 259*.

See also

"BestPerfGenPropDefaultSet" on page 88
"BestPerfGenPropGet" on page 89

BestPerfRun

Call

```
b_errtype BestPerfRun( b_handletype handle );
```

Description Starts all performance counters (and the trigger I/O sequencer).

CLI Equivalent `BestPerfRun`

CLI Abbreviation `prun`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestPerfStop" on page 97*
"BestTrigIORun" on page 177

BestPerfSeqProg

Call

```
b_errtype BestPerfSeqProg(
      b_handletype   handle,
      b_int32        measure );
```

Description Writes the information stored in the description table of the performance measure sequencer to the sequencer memory of the selected measure.

This function also checks whether the transition conditions of one state are consistent. If they are not, the function returns an error.

CLI Equivalent `BestPerfSeqProg measure=<measure>`

CLI Abbreviation `psprog meas=<measure>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

measure Value to identify the measure; see *"Measure Identifier" on page 87*.

See also –

BestPerfSeqPropDefaultSet

Call

```
b_errtype BestPerfSeqPropDefaultSet(
      b_handletype   handle,
      b_int32        measure );
```

Description

Initializes the sequencer description table of the selected measure.

For a description of properties and default values, refer to *"b_perfseqtranproptype" on page 260* and *"b_perfseqtrancondproptype" on page 259*.

CLI Equivalent

`BestPerfSeqPropDefaultSet measure=<measure>`

CLI Abbreviation

`psprpdefset meas=<measure>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

measure Value to identify the measure; see *"Measure Identifier" on page 87*.

See also

"BestPerfSeqTranPropDefaultSet" on page 94

BestPerfSeqTranCondPropSet

Call

```
b_errtype BestPerfSeqTranCondPropSet(
      b_handletype                handle,
      b_int32                     measure,
      b_int32                     transient,
      b_perfseqtrancondproptype   perfseqtrancondprop,
      b_charptrtype               condition);
```

Description

Sets a condition in the sequencer description table.

The conditions are transition condition, conditions to increment nominator or denominator counter, and conditions to decrement or preload the feedback counter.

CLI Equivalent

```
BestPerfSeqTranCondPropSet measure=<measure> transient=<transient>
perfseqtrancondprop=<perfseqtrancondprop> condition=<condition>
```

CLI Abbreviation

```
pstrancprpset meas=<measure> tran=<transient>
prop=<perfseqtrancondprop> con=<condition>
```

Return Value

Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

measure Value to identify the measure; see *“Measure Identifier” on page 87*.

transient Transient number (0 … 255).

perfseqtrancondprop Property to be set; see *“b_perfseqtrancondproptype” on page 259*.

condition Condition string to which the property is set. The string must be set in quotation marks. See *“Conditions Reference” on page 74*.

See also

“BestPerfSeqTranPropDefaultSet” on page 94
“BestPerfSeqTranPropSet” on page 95

BestPerfSeqTranPropDefaultSet

Call

```
b_errtype BestPerfSeqTranPropDefaultSet(
      b_handletype   handle,
      b_int32        measure,
      b_int32        transient );
```

Description

Sets all properties of a transient in the sequencer description table to default values.

For a description of properties and default values, refer to *"b_perfseqtranproptype" on page 260* and *"b_perfseqtrancondproptype" on page 259*.

CLI Equivalent

```
BestPerfSeqTranPropDefaultSet measure=<measure>
transient=<transient>
```

CLI Abbreviation

```
pstranprpdefset meas=<measure> tran=<transient>
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

measure Value to identify the measure; see *"Measure Identifier" on page 87*.

transient Transient number (0 … 255).

See also

"BestPerfSeqTranCondPropSet" on page 93
"BestPerfSeqTranPropSet" on page 95
"BestTrigSeqPropDefaultSet" on page 72

BestPerfSeqTranPropSet

Call

```
b_errtype BestPerfSeqTranPropSet(
      b_handletype           handle,
      b_int32                measure,
      b_int32                transient,
      b_perfseqtranproptype  perfseqtranprop,
      b_int32                value );
```

Description Sets a numeric transition property (“state” or “next state”) in the sequencer description table of the selected measure.

CLI Equivalent

```
BestPerfSeqTranPropSet measure=<measure> transient=<transient>
perfseqtranprop=<perfseqtranprop> value=<value>
```

CLI Abbreviation

```
pstranprpset meas=<measure> tran=<transient> prop=<perfseqtranprop>
val=<value>
```

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

measure Value to identify the measure; see *“Measure Identifier” on page 87*.

transient Transient number (0 … 255).

perfseqtranprop Property to be set; see *“b_perfseqtranproptype” on page 260*.

value Value to which the property is set.

See also

“BestPerfSeqTranCondPropSet” on page 93
“BestPerfSeqTranPropDefaultSet” on page 94

BestPerfStatusGet

Call

```
b_errtype BestPerfStatusGet(
      b_handletype   handle,
      b_int32        measure,
      b_int32        *value );
```

Description Queries the status of the selected measure.

CLI Equivalent `BestPerfStatusGet measure=<measure>`

CLI Abbreviation `psget meas=<measure>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

measure Value to identify the measure; see *"Measure Identifier" on page 87*.

Output Parameters **value** Value of the performance status register; see table below.

See also –

Performance Status Register

The following table shows the meanings of the single bits of the performance status register.

Bit	Meaning
0	1 = counter A overflow
1	1 = counter B overflow
2	1 = PCI clock reference counter overflow
3	1 = measure started
4 … 31	Not used.

BestPerfStop

Call `b_errtype BestPerfStop( b_handletype handle );`

Description Stops all performance counters (and the trigger I/O sequencer). Note that there is no need to stop the counters to get their current values.

CLI Equivalent `BestPerfStop`

CLI Abbreviation `pstop`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestPerfRun" on page 91*
"BestTrigIOStop" on page 180

BestPerfUpdate

Call `b_errtype BestPerfUpdate( b_handletype handle );`

Description Captures the values of all performance counters at the same PCI clock cycle. The counters are then reset and restarted.

The captured values can be read with *"BestPerfCtrRead" on page 87*.

CLI Equivalent `BestPerfUpdate`

CLI Abbreviation `pupdate`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also –

PCI Exerciser Functions

The PCI Exerciser functions are divided into the following sections:

- *"Exerciser Generic Functions" on page 100*
- *"Master Programming Functions" on page 111*
- *"Target Programming Functions" on page 127*
- *"Configuration Space Programming Functions" on page 144*
- *"Expansion ROM Programming Functions" on page 150*
- *"Data Memory Functions" on page 152*
- *"Host Access Functions" on page 155*
- *"Interrupt Generation Function" on page 163*
- *"Built-In Test Functions" on page 164*
- *"Hot Swap Debugging Functions" on page 169*

NOTE All Exerciser functions are only available with Opt. 300 of the Agilent E2920 series.

Exerciser Generic Functions

The following functions are used to prepare for exerciser programming:

Function	Result
"BestExerciserGenPropSet" on page 102	Sets the type of attribute memory organization.
"BestExerciserGenPropGet" on page 101	Reads the type of attribute memory organization.
"BestMasterBlockRun" on page 104	Runs one block with the transaction properties in the preparation register.
"BestMasterBlockPageRun" on page 103	Runs a block page of the master block transfer memory.
"BestMasterStop" on page 108	Stops the master.
"BestMasterGenPropDefaultSet" on page 105	Sets all master generic properties to default values.
"BestMasterGenPropSet" on page 107	Sets value of a master generic property.
"BestMasterCondStartPattSet" on page 108	Sets a master run condition.
"BestMasterGenPropGet" on page 106	Reads value of a master generic property.
"BestTargetGenPropDefaultSet" on page 109	Sets all target generic properties to default values.
"BestTargetGenPropSet" on page 110	Sets value of a target generic property.
"BestTargetGenPropGet" on page 109	Reads value of a target generic property.

How to use the functions is described in *"Programming the Exerciser"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestExerciserGenPropGet

Call

```
b_errtype BestExerciserGenPropGet(
      b_handletype             handle,
      b_exercisergenproptype   exeprop,
      b_int32                  *value );
```

Description Reads the generic exerciser property. The generic exerciser property determines the type of attribute memory organization.

CLI Equivalent `BestExerciserGenPropGet exeprop=<exeprop>`

CLI Abbreviation `egprpget prop=<exeprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

exeprop Exerciser property to be read; see *"b_exercisergenproptype" on page 247*.

Output Parameters **value** Value of the exerciser property; see *"b_exercisergenproptype" on page 247*.

See also *"BestExerciserGenPropSet" on page 102*

BestExerciserGenPropSet

Call

```
b_errtype BestExerciserGenPropSet(
      b_handletype            handle,
      b_exercisergenproptype  exeprop,
      b_int32                 value );
```

Description Sets the generic exerciser property. The generic exerciser property determines the type of attribute memory organization.

To make the setting persistent, use *“BestAllPropStore” on page 34*.

CLI Equivalent `BestExerciserGenPropSet exeprop=<exeprop> value=<value>`

CLI Abbreviation `egprpset prop=<exeprop> val=<value>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

exeprop Exerciser property to be set; see *“b_exercisergenproptype” on page 247*.

value Value to which the exerciser property is set; see *“b_exercisergenproptype” on page 247*.

See also *“BestExerciserGenPropGet” on page 101*

BestMasterBlockPageRun

Call

```
b_errtype BestMasterBlockPageRun(
      b_handletype   handle,
      b_int32        page_num );
```

Description Runs a block page in the master block transfer memory.

The function returns as soon as the block transfer has started. To ensure that a page run has been completed, poll the status of the master with *"BestStatusRegGet" on page 40* until the master is inactive.

Execution stops if a master abort occurs and a compare error is generated if the compare flag is set.

The master enable bit in the testcard's configuration space must be set to 1 before a master run is started. This bit can be set with *"BestMasterGenPropSet" on page 107*.

NOTE BestMasterBlockPageRun fails if you call this function while a transaction is running.

CLI Equivalent `BestMasterBlockPageRun page_num=<page_num>`

CLI Abbreviation `mbpgrun  page=<page_num>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

page_num Number of the block page that is to be executed (0 … 15, page 0 contains default values). All blocks defined in this page are executed consecutively.

See also *"BestMasterBlockRun" on page 104*

BestMasterBlockRun

Call

```
b_errtype BestMasterBlockRun( b_handletype handle );
```

Description

Runs one PCI block specified by the current settings in the preparation register of the master block transfer memory.

This function returns as soon as the block transfer has started. To ensure that a block run has been completed, poll the status of the master using *"BestStatusRegGet" on page 40* until the master is inactive.

Execution stops if a master abort occurs and a compare error is generated if the compare flag is set.

The master enable bit in the testcard's configuration space must be set to 1 before a master run is started. This bit can be set with *"BestMasterGenPropSet" on page 107*.

NOTE BestMasterBlockRun fails if you call this function while a transaction is running.

CLI Equivalent `BestMasterBlockRun`

CLI Abbreviation `mbrun`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle	Handle to identify the session.

See also *"BestMasterBlockPageRun" on page 103*

BestMasterGenPropDefaultSet

Call `b_errtype BestMasterGenPropDefaultSet( b_handletype handle );`

Description Sets all master generic properties to their default values. For a list of these properties, see *"b_mastergenproptype" on page 248*.

NOTE BestMasterGenPropDefaultSet fails if you call this function while a transaction is running.

Properties that affect settings of configuration space header registers will be altered only if the power up property B_PU_CONFRESTORE is set to 0. This property can be set with *"BestPowerUpPropSet" on page 38*.

CLI Equivalent `BestMasterGenPropDefaultSet`

CLI Abbreviations `mgprpdefset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestMasterGenPropSet" on page 107*
"BestMasterGenPropGet" on page 106

BestMasterGenPropGet

Call

```
b_errtype BestMasterGenPropGet(
      b_handletype         handle,
      b_mastergenproptype  mastergenprop,
      b_int32              *value );
```

Description Reads the value of a generic master property.

NOTE BestMasterGenPropGet fails if you call this function while a transaction is running.

CLI Equivalent `BestMasterGenPropGet mastergenprop=<mastergenprop>`

CLI Abbreviation `mgprpget prop=<mastergenprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

mastergenprop Property to be read; see *"b_mastergenproptype" on page 248*.

Output Parameters

value Property value; see *"b_mastergenproptype" on page 248*.

See also

"BestMasterGenPropDefaultSet" on page 105
"BestMasterGenPropSet" on page 107

BestMasterGenPropSet

Call

```
b_errtype BestMasterGenPropSet(
      b_handletype          handle,
      b_mastergenproptype   mastergenprop,
      b_int32               value );
```

Description Sets the value of a generic master property. Generic master properties are valid during a master run.

NOTE BestMasterGenPropSet fails if you call this function while a transaction is running.

CLI Equivalent `BestMasterGenPropSet mastergenprop=<mastergenprop> value=<value>`

CLI Abbreviations `mgprpset prop=<mastergenprop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

mastergenprop Property to be set; see *"b_mastergenproptype" on page 248*.

value Value to which the property is set; see *"b_mastergenproptype" on page 248*.

See also *"BestMasterGenPropDefaultSet" on page 105*
"BestMasterGenPropGet" on page 106

BestMasterCondStartPattSet

Call

```
b_errtype BestMasterCondStartPattSet(
      b_handletype    handle,
      b_charptrtype   pattern );
```

Description Sets condition for the start of the master. This allows you to conditionally start the master based on a specific bus event. The bus events (bus signals) can constitute the master start condition.

For a list of the available bus signals, refer to the *“b_signaltype (List of Signals)” on page 265*.

NOTE BestMasterCondStartPattSet fails if you call this function while a transaction is running.

CLI Equivalent `BestMasterCondStartPattSet pattern=<pattern>`

CLI Abbreviation `mcspset patt=<pattern>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

pattern The pattern string that defines the compare pattern. Please refer to *“BestPattSet” on page 64* for details on pattern term programming and pattern syntax.

See also –

BestMasterStop

Call `b_errtype BestMasterStop( b_handletype handle );`

Description Stops the master after completing the current data transfer.

CLI Equivalent `BestMasterStop`

CLI Abbreviation `mstop`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

See also –

BestTargetGenPropDefaultSet

Call
```
b_errtype BestTargetGenPropDefaultSet ( b_handletype handle );
```

Description Sets the target generic properties to their default values. For a list of these properties, see *"b_targetgenproptype" on page 276*.

NOTE Properties that affect settings of configuration space header registers will be altered only if the power up property B_PU_CONFRESTORE is set to 0. This property can be set with *"BestPowerUpPropSet" on page 38*.

CLI Equivalent `BestTargetGenPropDefaultSet`

CLI Abbreviation `tgprpdefset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also
"BestTargetGenPropSet" on page 110
"BestTargetGenPropGet" on page 109

BestTargetGenPropGet

Call
```
b_errtype BestTargetGenPropGet(
      b_handletype         handle,
      b_targetgenproptype  targetgenprop,
      b_int32              *value );
```

Description Reads a generic target property.

CLI Equivalent `BestTargetGenPropGet targetgenprop=<targetgenprop>`

CLI Abbreviation `tgprpget prop=<targetgenprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

targetgenprop Property to be read; see *"b_targetgenproptype" on page 276*.

Output Parameters **value** Property value; see *"b_targetgenproptype" on page 276*.

See also
"BestTargetGenPropDefaultSet" on page 109
"BestTargetGenPropSet" on page 110

BestTargetGenPropSet

Call

```
b_errtype BestTargetGenPropSet(
      b_handletype         handle,
      b_targetgenproptype  targetgenprop,
      b_int32              value );
```

Description Sets a generic target property.

CLI Equivalent `BestTargetGenPropSet targetgenprop=<targetgenprop> value=<value>`

CLI Abbreviation `tgprpset prop=<targetgenprop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

targetgenprop Property to be set; see *"b_targetgenproptype" on page 276*.

value Value to which the property is set; see *"b_targetgenproptype" on page 276*.

See also

"BestTargetGenPropDefaultSet" on page 109
"BestTargetGenPropGet" on page 109

Master Programming Functions

Master Block Transfer Pages

The following functions are used to program the master block transfer pages:

Function	Result
"BestMasterBlockPageInit" on page 123	Initializes a master block transfer memory page.
"BestMasterBlockPropDefaultSet" on page 124	Sets the master block transfer memory's preparation register to defaults.
"BestMasterBlockPropSet" on page 125	Sets a master block transaction property in the preparation register.
"BestMasterBlockPropGet" on page 124	Reads a master block transaction property from the preparation register.
"BestMasterBlockLineProg" on page 121	Programs the preparation register to a master block transfer memory line.
"BestMasterBlockLineRead" on page 122	Reads a master block transfer memory line to the preparation register.
"BestMasterBlockEndProg" on page 120	Programs an "end of page" (EOP) to a master block transfer memory line.
"BestMasterBlockPageInit" on page 123	Initializes a master block transfer memory page.

Master Attribute Pages

The following functions are used to program master attribute pages:

Function	Result
"BestMasterAttrPageInit" on page 117	Initializes a master attribute memory page.
"BestMasterAttrPropDefaultSet" on page 118	Sets the master attribute memory's preparation register to defaults.
"BestMasterAttrPropSet" on page 119	Sets a master attribute property in the preparation register.
"BestMasterAttrPropGet" on page 118	Reads a master attribute property from the preparation register.
"BestMasterAttrLineProg" on page 115	Programs the master attribute memory's preparation register to a memory line.
"BestMasterAttrLineRead" on page 116	Reads a master attribute memory line to the preparation register.
"BestMasterAttrGroupLineProg" on page 113	Programs a group of master attributes from the preparation register to a memory line.
"BestMasterAttrGroupLineRead" on page 114	Reads a group of master attributes from a memory line to the preparation register.

Byte Enable Memory

The following functions are used to program the byte enable memory:

Function	Result
"BestMasterByteEnableProg" on page 126	Programs a line in the byte enable memory.
"BestMasterByteEnableRead" on page 126	Reads a line from the byte enable memory.

How to use the functions is described in *"Programming the Exerciser as a Master Device"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestMasterAttrGroupLineProg

Call

```
b_errtype BestMasterAttrGroupLineProg(
      b_handletype       handle,
      b_mattrgrouptype   group,
      b_int32            page_num,
      b_int32            offset );
```

Description

Writes the attributes of a group from the preparation register to a line in the attribute memory.

After you have programmed all memory lines within a group, set the individual loop bit B_M_DOLOOP with *"BestMasterAttrPropSet" on page 119* to indicate the last line of the current group.

NOTE Within a page you can use either group or non-group memory programming functions, but not both together.

CLI Equivalent

```
BestMasterAttrGroupLineProg group=<group> page_num=<page_num>
offset=<offset>
```

CLI Abbreviation

```
magrplprog grp=<group> page=<page_num> offs=<offset>
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

group Identifier of the group; see *"b_mattrgrouptype" on page 250*.

page_num Number of the page on which the line is to be programmed.

Valid page numbers are:

- 1 … 63 if page size is 4
- 1 … 7 if page size is 32

The page size is set by the generic exerciser property B_EGEN_ATTRPAGESIZE with *"BestExerciserGenPropSet" on page 102*.

offset Number of the line to be programmed. Line counting starts from 0.

NOTE You can program more lines than are available on a page. This results in concatenated pages.

See also

"BestMasterAttrPageInit" on page 117
"BestMasterAttrGroupLineRead" on page 114
"BestMasterAttrLineProg" on page 115

BestMasterAttrGroupLineRead

Call

```
b_errtype BestMasterAttrGroupLineRead(
      b_handletype       handle,
      b_mattrgrouptype   group,
      b_int32            page_num,
      b_int32            offset );
```

Description Reads the attributes of a line within a group in the attribute memory to the preparation register.

CLI Equivalent

```
BestMasterAttrGroupLineRead group=<group> page_num=<page_num>
offset=<offset>
```

CLI Abbreviation

```
magrplread grp=<group> page=<page_num> offs=<offset>
```

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

group Identifier of the group; see *"b_mattrgrouptype" on page 250*.

page_num Number of the page on which the line is to be read.

offset Number of the line to be read. Line counting starts from 0.

See also
"BestMasterAttrPageInit" on page 117
"BestMasterAttrGroupLineProg" on page 113
"BestMasterAttrLineRead" on page 116

BestMasterAttrLineProg

Call

```
b_errtype BestMasterAttrLineProg(
      b_handletype   handle,
      b_int32        page_num,
      b_int32        offset );
```

Description Writes the contents of the preparation register to a line in the attribute memory.

After you have programmed all memory lines, set the general loop bit B_M_DOLOOP with *"BestMasterAttrPropSet" on page 119* to indicate the last line in the attribute page.

NOTE Within a page you can use either group or non-group memory programming functions, but not both together.

CLI Equivalent `BestMasterAttrLineProg page_num=<page_num> offset=<offset>`

CLI Abbreviation `malprog page=<page_num> offs=<offset>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

page_num Number of the page on which the line is to be programmed.

Valid page numbers are:

- 1 … 63 if page size is 4
- 1 … 7 if page size is 32

The page size is set by the generic exerciser property B_EGEN_ATTRPAGESIZE with *"BestExerciserGenPropSet" on page 102*.

offset Number of the line to be programmed. Line counting starts from 0.

NOTE You can program more lines than are available on a page. This results in concatenated pages.

See also *"BestMasterAttrPageInit" on page 117*
"BestMasterAttrLineRead" on page 116
"BestMasterAttrGroupLineProg" on page 113

BestMasterAttrLineRead

Call

```
b_errtype BestMasterAttrLineRead(
      b_handletype   handle,
      b_int32        page_num,
      b_int32        offset );
```

Description Reads the contents of an attribute memory's line to the preparation register.

CLI Equivalent `BestMasterAttrLineRead page_num=<page_num> offset=<offset>`

CLI Abbreviation `malread page=<page_num> offs=<offset>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

page_num Number of the page on which the line is to be read.

offset Number of the line to be read. Line counting starts from 0.

See also

"BestMasterAttrPageInit" on page 117
"BestMasterAttrLineProg" on page 115
"BestMasterAttrGroupLineRead" on page 114

BestMasterAttrPageInit

Call

```
b_errtype BestMasterAttrPageInit(
      b_handletype   handle,
      b_int32        page_num );
```

Description

Initializes a master attribute memory page (and all pages concatenated to this page).

This function must be called once before an attribute page can be programmed or read.

NOTE BestMasterAttrPageInit fails if you call this function while a transaction is running.

CLI Equivalent `BestMasterAttrPageInit page_num=<page_num>`

CLI Abbreviation `mapginit page=<page_num>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

page_num Number of the memory page to be initialized.

Valid page numbers are:

- 1 … 63 if page size is 4
- 1 … 7 if page size is 32

The page size is set by the generic exerciser property B_EGEN_ATTRPAGESIZE with *“BestExerciserGenPropSet” on page 102*.

See also

“BestMasterAttrGroupLineProg” on page 113
“BestMasterAttrGroupLineRead” on page 114
“BestMasterAttrLineProg” on page 115
“BestMasterAttrLineRead” on page 116

BestMasterAttrPropDefaultSet

Call

```
b_errtype BestMasterAttrPropDefaultSet( b_handletype handle );
```

Description Sets the preparation register of the master attribute memory to default values. For a description of attributes and default values, see *"b_mattrproptype" on page 251*.

CLI Equivalent `BestMasterAttrPropDefaultSet`

CLI Abbreviation `maprpdefset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestMasterAttrPropSet" on page 119*
"BestMasterAttrPropGet" on page 118

BestMasterAttrPropGet

Call

```
b_errtype BestMasterAttrPropGet(
      b_handletype      handle,
      b_mattrproptype   mattrprop,
      b_int32           *value );
```

Description Reads a master attribute from the preparation register.

CLI Equivalent `BestMasterAttrPropGet mattrprop=<mattrprop>`

CLI Abbreviation `maprpget prop=<mattrprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

mattrprop Master attribute to be read; see *"b_mattrproptype" on page 251*.

Output Parameters **value** Value of the attribute; see *"b_mattrproptype" on page 251*.

See also *"BestMasterAttrPropDefaultSet" on page 118*
"BestMasterAttrPropSet" on page 119

BestMasterAttrPropSet

Call

```
b_errtype BestMasterAttrPropSet(
      b_handletype      handle,
      b_mattrproptype   mattrprop,
      b_int32           value );
```

Description Sets a master attribute in the preparation register.

After you have set all attributes to the required values in the register, you can program the complete register to a memory line with *"BestMasterAttrLineProg" on page 115*.

CLI Equivalent `BestMasterAttrPropSet  mattrprop=<mattrprop> value=<value>`

CLI Abbreviation `maprpset prop=<mattrprop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

mattrprop Master attribute to be set; see *"b_mattrproptype" on page 251*.

value Value to which the attribute is set; see *"b_mattrproptype" on page 251*.

See also *"BestMasterAttrPropDefaultSet" on page 118*
"BestMasterAttrPropGet" on page 118

BestMasterBlockEndProg

Call

```
b_errtype BestMasterBlockEndProg(
      b_handletype   handle,
      b_int32        page_num,
      b_int32        offset );
```

Description Programs the “end of page” (EOP) line in the master block transfer memory. This function should conclude memory programming.

NOTE After initialization, the last line of a page is pre-programmed with EOP.

CLI Equivalent `BestMasterBlockEndProg page_num=<page_num> offset=<offset>`

CLI Abbreviation `mbeprog page=<page_num> offs=<offset>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

page_num Number of the block transfer memory page.

offset Number of the line to be programmed. Line counting starts from 0.

See also *“BestMasterBlockLineProg” on page 121*

BestMasterBlockLineProg

Call

```
b_errtype BestMasterBlockLineProg(
      b_handletype   handle,
      b_int32        page_num
      b_int32        offset );
```

Description

Writes the preparation register to a line of a master block transfer memory page.

After you have programmed all required memory lines, program the “end of page” to your last memory line with *“BestMasterBlockEndProg” on page 120* (this is not necessary if the last line is physically the end of the page).

NOTE

The last line in the memory cannot be programmed, it can only contain an EOP.

CLI Equivalent

`BestMasterBlockLineProg page_num=<page_num> offset=<offset>`

CLI Abbreviation

`mblprog page=<page_num> offs=<offset>`

Return Value

Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

page_num Number of the block transfer memory page (1 … 15, page 0 ist read-only).

Pages can be unavailable if they are part of a sequence of concatenated pages (if the physical page size is overridden).

offset Number of the line to be programmed. Line counting starts from 0.

See also

“BestMasterBlockPageInit” on page 123
“BestMasterBlockLineRead” on page 122

BestMasterBlockLineRead

Call

```
b_errtype BestMasterBlockLineRead(
      b_handletype   handle,
      b_int32        page_num,
      b_int32        offset,
      b_int32        *eop );
```

Description

Reads the properties of one block (that is: one line of the master block transfer memory) to the preparation register, if this line is not an EOP line.

If the line is an EOP line, the preparation register remains unchanged and the EOP flag (output parameter) is set to 1.

CLI Equivalent

`BestMasterBlockLineRead page_num=<page_num> offset=<offset>`

CLI Abbreviation

`mblread page=<page_num> offs=<offset>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

page_num Number of the block transfer memory page.

offset Number of the line to be read. Line counting starts from 0.

Output Parameters

eop EOP flag:

- 0 – the read line is not an EOP line.
- 1 – the read line is an EOP line.

See also

"BestMasterBlockLineProg" on page 121

BestMasterBlockPageInit

Call

```
b_errtype BestMasterBlockPageInit(
      b_handletype   handle,
      b_int32        page_num );
```

Description Initializes a master block transfer page and sets the current block pointer to the beginning of the page. This function must be called once before a page can be programmed.

If this function is applied to a concatenated page, all concatenated pages—previous and next—will also be initialized.

NOTE MasterBlockPageInit fails if you call this function while a transaction is running.

CLI Equivalent `BestMasterBlockPageInit page_num=<page_num>`

CLI Abbreviation `mbpginit page=<page_num>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

page_num Number of the page to be initialized (1 … 15, page 0 cannot be overwritten).

See also *"BestMasterBlockLineProg" on page 121*

BestMasterBlockPropDefaultSet

Call
```
b_errtype BestMasterBlockPropDefaultSet( b_handletype handle );
```

Description Sets the preparation register of the block transfer memory to default values.

For a description of properties and default values, see *“b_blkproptype” on page 228*.

CLI Equivalent `BestMasterBlockPropDefaultSet`

CLI Abbreviation `mbprpdefset`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *“BestMasterBlockPropSet” on page 125*
“BestMasterBlockPropGet” on page 124

BestMasterBlockPropGet

Call
```
b_errtype BestMasterBlockPropGet(
      b_handletype    handle,
      b_blkproptype   blk_prop,
      b_int32         *value );
```

Description Reads a master block transaction property from the preparation register.

CLI Equivalent `BestMasterBlockPropGet blk_prop=<blk_prop>`

CLI Abbreviation `mbprpget prop=<blk_prop>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

blk_prop Block transaction property to be read; see *“b_blkproptype” on page 228*.

Output Parameters **value** Value of the property; see *“b_blkproptype” on page 228*.

See also *“BestMasterBlockPropDefaultSet” on page 124*
“BestMasterBlockPropSet” on page 125

BestMasterBlockPropSet

Call

```
b_errtype BestMasterBlockPropSet(
      b_handletype    handle,
      b_blkproptype   blk_prop,
      b_int32         value );
```

Description Sets a master block transaction property in the preparation register.

After you have set all properties to the required values in the register, the complete register can be programmed to a memory line with *"BestMasterBlockLineProg" on page 121*.

CLI Equivalent `BestMasterBlockPropSet blk_prop=<blk_prop> value=<value>`

CLI Abbreviation `mbprpset prop=<blk_prop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

blk_prop Property to be set; see *"b_blkproptype" on page 228*.

value Value to which the property is set; see *"b_blkproptype" on page 228*.

See also

"BestMasterBlockPropDefaultSet" on page 124
"BestMasterBlockPropGet" on page 124

BestMasterByteEnableProg

Call

```
b_errtype BestMasterByteEnableProg(
      b_handletype   handle,
      b_int32        offset,
      b_int32        value );
```

Description Programs the byte enables for one transaction to the byte enable memory.

CLI Equivalent `BestMasterByteEnableProg offset=<offset> value=<value>`

CLI Abbreviation `mbytenprog offs=<offset> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

offset Byte enable memory line. Programmable lines are 16 … 255 (lines 0 … 15 are read-only for compatibility reasons).

value Value (0 … 255) to be programmed to the byte enable memory line.

See also *"BestMasterByteEnableRead" on page 126*

BestMasterByteEnableRead

Call

```
b_errtype BestMasterByteEnableRead(
      b_handletype   handle,
      b_int32        offset,
      b_int32        *value );
```

Description Reads one line from byte enable memory (byte enables for one transfer).

CLI Equivalent `BestMasterByteEnableRead offset=<offset>`

CLI Abbreviation `mbytenread offs=<offset>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

offset Byte enable memory line to be read (0 … 255).

Output Parameters

value Value programmed in the referring byte enable memory line.

See also *"BestMasterByteEnableProg" on page 126*

Target Programming Functions

The functions described in this section are used to program the exerciser as a target.

Decoder Properties

The following functions are used to program the decoder properties:

Function	Result
"BestTargetDecoderPropSet" on page 142	Sets a decoder property in the preparation register.
"BestTargetDecoderPropGet" on page 141	Reads a decoder property from the preparation register.
"BestTargetDecoderProg" on page 139	Programs a decoder with properties as set in the preparation register.
"BestTargetDecoderRead" on page 143	Reads the properties of a decoder to the preparation register.
"BestTargetDecoderPowerUpProg" on page 137	Programs the power up properties of a decoder with properties as set in the preparation register.
"BestTargetDecoderPowerUpRead" on page 138	Reads the power up properties of a decoder to the preparation register.

Target Attribute Memory

The following functions are used to program the target attribute memory:

Function	Result
"BestTargetAttrPageInit" on page 133	Initializes a target attribute memory page.
"BestTargetAttrPropDefaultSet" on page 134	Sets the target attribute memory's preparation register to the default values.
"BestTargetAttrPropSet" on page 136	Sets a target attribute property in the preparation register.
"BestTargetAttrPropGet" on page 135	Reads a target attribute property from the preparation register.
"BestTargetAttrLineProg" on page 131	Programs the target attribute memory's preparation register to a memory line.
"BestTargetAttrLineRead" on page 132	Reads a target attribute memory line to the preparation register.
"BestTargetAttrGroupLineProg" on page 129	Programs a group of target attributes from the preparation register to a memory line.
"BestTargetAttrGroupLineRead" on page 130	Reads a group of target attributes from a memory line to the preparation register.
"BestTargetAttrPageSelect" on page 134	Selects a target attribute memory page for transfer.

How to use the functions is described in "*Programming the Exerciser as a Target Device*" in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestTargetAttrGroupLineProg

Call

```
b_errtype BestTargetAttrGroupLineProg(
      b_handletype       handle,
      b_tattrgrouptype   group,
      b_int32            page_num,
      b_int32            offset );
```

Description Writes the attributes of a group from the preparation register to a line in the attribute memory.

After you have programmed all memory lines within a group, set the individual loop bit B_T_DOLOOP with *"BestTargetAttrPropSet" on page 136* to indicate the last line of the current group.

NOTE Within a page you can use either group or non-group memory programming functions, but not both together.

CLI Equivalent `BestTargetAttrGroupLineProg group=<group> page_num=<page_num> offset=<offset>`

CLI Abbreviation `tagrplprog grp=<group> page=<page_num> offs=<offset>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

group Identifier of the group; see *"b_tattrgrouptype" on page 277*.

page_num Number of the page on which the line is to be programmed.

Valid page numbers are:

- 1 … 63 if page size is 4
- 1 … 7 if page size is 32

The page size is set by the generic exerciser property B_EGEN_ATTRPAGESIZE with *"BestExerciserGenPropSet" on page 102*.

offset Number of the line to be programmed. Line counting starts from 0.

NOTE You can program more lines than are available on a page. This results in concatenated pages.

See also
"BestTargetAttrPageInit" on page 133
"BestTargetAttrGroupLineRead" on page 130
"BestTargetAttrLineProg" on page 131

BestTargetAttrGroupLineRead

Call

```
b_errtype BestTargetAttrGroupLineRead(
      b_handletype       handle,
      b_tattrgrouptype   group,
      b_int32            page_num,
      b_int32            offset );
```

Description

Reads the attributes of a group from a line in the attribute memory to the preparation register.

CLI Equivalent

```
BestTargetAttrGroupLineRead group=<group> page_num=<page_num>
offset=<offset>
```

CLI Abbreviation

```
tagrplread grp=<group> page=<page_num> offs=<offset>
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

group Identifier of the group; see *"b_tattrgrouptype" on page 277*.

page_num Number of the page on which the line is to be read.

Valid page numbers are:

- 1 … 63 if page size is 4
- 1 … 7 if page size is 32

The page size is set by the generic exerciser property B_EGEN_ATTRPAGESIZE with *"BestExerciserGenPropSet" on page 102*.

offset Number of the line to be read. Line counting starts from 0.

See also

"BestTargetAttrPageInit" on page 133
"BestTargetAttrGroupLineProg" on page 129
"BestTargetAttrLineRead" on page 132

BestTargetAttrLineProg

Call

```
b_errtype BestTargetAttrLineProg(
      b_handletype   handle,
      b_int32        page_num,
      b_int32        offset );
```

Description

Writes the contents of the preparation register to a line in the attribute memory.

After you have programmed all memory lines, set the general loop bit B_T_DOLOOP with *"BestTargetAttrPropSet" on page 136* to indicate the last line in the attribute page.

NOTE Within a page you can use either group or non-group memory programming functions, but not both together.

CLI Equivalent

```
BestTargetAttrLineProg page_num=<page_num> offset=<offset>
```

CLI Abbreviation

```
talprog page=<page_num> offs=<offset>
```

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

page_num Number of the page on which the line is to be programmed.

Valid page numbers are:

- 1 … 63 if page size is 4
- 1 … 7 if page size is 32

The page size is set by the generic exerciser property B_EGEN_ATTRPAGESIZE with *"BestTargetAttrPropSet" on page 136*.

offset Number of the line to be programmed. Line counting starts from 0.

NOTE You can program more lines than are available on a page. This results in concatenated pages.

See also

"BestTargetAttrPageInit" on page 133
"BestTargetAttrGroupLineProg" on page 129
"BestTargetAttrLineRead" on page 132

BestTargetAttrLineRead

Call

```
b_errtype BestTargetAttrLineRead(
      b_handletype   handle,
      b_int32        page_num,
      b_int32        offset );
```

Description Reads the contents of an attribute memory's line to the preparation register.

CLI Equivalent `BestTargetAttrLineRead page_num=<page_num> offset=<offset>`

CLI Abbreviation `talread page=<page_num> offs=<offset>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

page_num Number of the page from which the line is to be read.

offset Number of the line to be read.

See also

"BestTargetAttrPageInit" on page 133
"BestTargetAttrGroupLineRead" on page 130
"BestTargetAttrLineProg" on page 131

BestTargetAttrPageInit

Call

```
b_errtype BestTargetAttrPageInit(
      b_handletype   handle,
      b_int32        page_num );
```

Description

Initializes a target attribute memory page (and all pages concatenated to this page).

This function must be called once before an attribute page can be programmed or read.

NOTE

BestTargetAttrPageInit fails if you call this function while a transaction is running.

CLI Equivalent

```
BestTargetAttrPageInit page_num=<page_num>
```

CLI Abbreviation

```
tapginit page=<page_num>
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

page_num Number of the memory page to be initialized.

Valid page numbers are:

- 1 … 63 if page size is 4
- 1 … 7 if page size is 32

The page size is set by the generic exerciser property B_EGEN_ATTRPAGESIZE with *"BestTargetAttrPropSet" on page 136*.

See also

"BestTargetAttrGroupLineProg" on page 129
"BestTargetAttrGroupLineRead" on page 130
"BestTargetAttrLineProg" on page 131
"BestTargetAttrLineRead" on page 132

BestTargetAttrPageSelect

Call

```
b_errtype BestTargetAttrPageSelect (
      b_handletype   handle,
      b_int32        page_num );
```

Description Selects a target attribute page.

CLI Equivalent `BestTargetAttrPageSelect page_num=<page_num>`

CLI Abbreviation `tapgsel page=<page_num>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

page_num Number of the memory page to be selected.

Valid page numbers are:

- 1 … 63 if page size is 4
- 1 … 7 if page size is 32

The page size is set by the generic exerciser property B_EGEN_ATTRPAGESIZE with *"BestTargetAttrPropSet" on page 136*.

See also –

BestTargetAttrPropDefaultSet

Call

```
b_errtype BestTargetAttrPropDefaultSet( b_handletype handle );
```

Description Sets the preparation register of the target attribute memory to default values. For a description of attributes and default values, see *"b_tattrproptype" on page 278*.

CLI Equivalent `BestTargetAttrPropDefaultSet`

CLI Abbreviation `taprpdefset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestMasterAttrPropSet" on page 119*
"BestMasterAttrPropGet" on page 118

BestTargetAttrPropGet

Call

```
b_errtype BestTargetAttrPropGet(
      b_handletype      handle,
      b_tattrproptype   tattrprop,
      b_int32           *value );
```

Description Reads a target attribute from the attribute preparation register.

CLI Equivalent `BestTargetAttrPropGet tattrprop=<tattrprop>`

CLI Abbreviation `taprpget prop=<tattrprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

tattrprop Target attribute to be read; see *"b_tattrproptype" on page 278*.

Output Parameters

value Value of the attribute; see *"b_tattrproptype" on page 278*.

See also

"BestMasterAttrPropDefaultSet" on page 118
"BestMasterAttrPropSet" on page 119

BestTargetAttrPropSet

Call

```
b_errtype BestTargetAttrPropSet(
      b_handletype      handle,
      b_tattrproptype   tattrprop,
      b_int32           value );
```

Description

Sets a target attribute in the preparation register of the target attribute memory.

After you have set all attributes to the required values in the register, you can program the complete register to a memory line with *"BestTargetAttrLineProg" on page 131*.

CLI Equivalent

`BestTargetAttrPropSet tattrprop=<tattrprop> value=<value>`

CLI Abbreviation

`taprpset prop=<tattrprop> val=<value>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

tattrprop Target attribute to be set; see *"b_tattrproptype" on page 278*.

value Value to which the attribute is set; see *"b_tattrproptype" on page 278*.

See also

"BestMasterAttrPropDefaultSet" on page 118
"BestMasterAttrPropGet" on page 118

BestTargetDecoderPowerUpProg

Call

```
b_errtype BestTargetDecoderPowerUpProg(
      b_handletype    handle,
      b_decodertype   decoder );
```

Description

Programs power up properties of a decoder with the properties set in the preparation register by means of *"BestTargetDecoderPropSet" on page 142*. (Use BestTargetDecoderPropSet to change decoder base address register settings or decoder size when the card is operating.)

The function also checks whether the property values in the target decoder preparation register are consistent with the specified decoder, except when the decoder is set to "custom" behavior.

NOTE BestTargetDecoderPowerUpProg fails if you call this function while a transaction is running.

CLI Equivalent `BestTargetDecoderPowerUpProg decoder_num=<decoder>`

CLI Abbreviation `tdpuprog dec=<decoder>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

decoder Decoder to be programmed; see *"b_decodertype" on page 234*.

See also

"BestTargetDecoderPowerUpRead" on page 138
"BestTargetDecoderProg" on page 139

BestTargetDecoderPowerUpRead

Call

```
b_errtype BestTargetDecoderPowerUpRead(
      b_handletype    handle,
      b_decodertype   decoder );
```

Description Reads back a set of decoder power up properties to the decoder property preparation register.

CLI Equivalent `BestTargetDecoderPowerUpRead decoder_num=<decoder>`

CLI Abbreviation `tdpuread dec=<decoder>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

decoder Decoder to be read; see *"b_decodertype" on page 234*.

See also *"BestTargetDecoderPowerUpProg" on page 137*
"BestTargetDecoderRead" on page 143

BestTargetDecoderProg

Call

```
b_errtype BestTargetDecoderProg(
      b_handletype    handle,
      b_decodertype   decoder );
```

Description Programs a decoder using the properties set in the preparation register.

This function also checks whether the property values in the target decoder preparation register are consistent with the specified decoder, except when the decoder is set to “custom” behavior.

NOTE BestTargetDecoderProg fails if you call this function while a transaction is running.

CLI Equivalent `BestTargetDecoderProg decoder_num=<decoder>`

CLI Abbreviation `tdprog  dec=<decoder>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

decoder Decoder to be programmed; see *“b_decodertype” on page 234*.

See also *“BestTargetDecoderPowerUpProg” on page 137*
“BestTargetDecoderRead” on page 143

BestTargetDecoderPropDefaultSet

Call

```
b_errtype BestTargetDecoderPropDefaultSet(
      b_handletype    handle,
      b_decodertype   decoder_num );
```

Description Sets the preparation register to the default values of a decoder. Each decoder provides a specific parameter set.

CLI Equivalent `BestTargetDecoderPropDefaultSet decoder_num=<decoder_num>`

CLI Abbreviation `tdprpdefset dec=<decoder_num>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

decoder_num Number of the decoder of which the default values are to be set in the preparation register. See *“b_decodertype” on page 234*.

See also *“BestTargetDecoderPropSet” on page 142*
“BestTargetDecoderPropGet” on page 141

BestTargetDecoderPropGet

Call

```
b_errtype BestTargetDecoderPropGet(
      b_handletype    handle,
      b_decproptype   decoder_prop,
      b_int32         *value );
```

Description Reads a decoder property from the decoder preparation register.

NOTE BestTargetDecoderPropGet fails if you call this function while a transaction is running.

CLI Equivalent `BestTargetDecoderPropGet decoder_prop=<decoder_prop>`

CLI Abbreviation `tdprpget prop=<decoder_prop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

decoder_prop Decoder property to be read; see *"b_decproptype" on page 234*.

Output Parameters

value Property value; see *"b_decproptype" on page 234*.

See also

"BestTargetDecoderPropDefaultSet" on page 140
"BestTargetDecoderPropSet" on page 142

BestTargetDecoderPropSet

Call

```
b_errtype BestTargetDecoderPropSet(
      b_handletype    handle,
      b_decproptype   decoder_prop,
      b_int32         value );
```

Description Sets a property of a decoder in the preparation register.

NOTE This function overrides configuration space settings made with *"BestConfRegSet" on page 146*.

CLI Equivalent `BestTargetDecoderPropSet decoder_prop=<decoder_prop> value=<value>`

CLI Abbreviation `tdprpset  prop=<decoder_prop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

decoder_prop Defines the property to be set; see *"b_decproptype" on page 234*.

value Value to which the specified property is set; see *"b_decproptype" on page 234*.

See also *"BestTargetDecoderPropDefaultSet" on page 140*
"BestTargetDecoderPropGet" on page 141

BestTargetDecoderRead

Call

```
b_errtype BestTargetDecoderRead(
      b_handletype    handle,
      b_decodertype   decoder );
```

Description Reads back a set of decoder properties to the decoder property preparation register.

CLI Equivalent `BestTargetDecoderRead decoder_num=<decoder>`

CLI Abbreviation `tdread  dec=<decoder>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

decoder Decoder to be read; see *"b_decodertype" on page 234*.

See also
"BestTargetDecoderProg" on page 139
"BestTargetDecoderPowerUpRead" on page 138

Configuration Space Programming Functions

The following functions program the configuration space:

Function	Result
"BestConfRegSet" on page 146	Sets a register in the configuration space.
"BestConfRegGet" on page 145	Reads a register in the configuration space.
"BestConfigScan" on page 149	Scans configuration space header registers.
"BestConfigScanPrint" on page 149	Returns configuration space header registers.
"BestConfRegMaskSet" on page 148	Sets a bit mask for read-only bits in a configuration space register.
"BestConfRegMaskGet" on page 147	Reads a bit mask for read-only bits in a configuration space register.
"BestPCIConfigCheck" on page 150	Checks the configuration space header settings for PCI compliance.

How to use the functions is described in *"Configuration Space Header Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestConfRegGet

Call

```
b_errtype BestConfRegGet(
      b_handletype   handle,
      b_int32        offset,
      b_int32        *value );
```

Description Reads a value of a configuration space register.

CLI Equivalent `BestConfRegGet offset=<offset>`

CLI Abbreviation `conrget offs=<offset>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

offset Address offset in the configuration space (00 … 3C\h, dword aligned).

Output Parameters **value** Value of the register (32-bit value).

See also *"BestConfRegSet" on page 146*

BestConfRegSet

Call

```
b_errtype BestConfRegSet(
      b_handletype   handle,
      b_int32        offset,
      b_int32        value );
```

Description Sets a register of the configuration space to the specified value.

NOTE BestConfRegSet fails if you call this function while a transaction is running or if the settings conflict with those of a decoder property or decoder behavior. Refer to *"b_decproptype" on page 234*.

CLI Equivalent `BestConfRegSet offset=<offset> value=<value>`

CLI Abbreviation `conrset offs=<offset> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

offset Address offset in the configuration space (00 … 3C\h, dword aligned).

value Value to which the register is set (32-bit value).

See also *"BestConfRegGet" on page 145*

BestConfRegMaskGet

Call

```
b_errtype BestConfRegMaskGet(
      b_handletype   handle,
      b_int32        offset,
      b_int32        *value );
```

Description Reads the mask that defines which bits in a configuration space register are set to read-only and which to read/write for configuration accesses.

CLI Equivalent `BestConfRegMaskGet offset=<offset>`

CLI Abbreviation `conrmaskget offs=<offset>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

offset Address offset in the configuration space (00 … 3C\h, dword aligned).

Output Parameters

value 32-bit mask:

- 0 – read-only
- 1 – read/write

See also *“BestConfRegMaskSet” on page 148*

BestConfRegMaskSet

Call

```
b_errtype BestConfRegMaskSet(
      b_handletype   handle,
      b_int32        offset,
      b_int32        value );
```

Description Sets the mask that defines which bits in a configuration space register are set to read-only and which to read/write for configuration accesses.

NOTE This function fails if the settings conflict with those of a decoder property or decoder behavior. Refer to *"b_decproptype" on page 234*.

CLI Equivalent `BestConfRegMaskSet offset=<offset> value=<value>`

CLI Abbreviation `conrmaskset offs=<offset> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

offset Address offset in the configuration space (00 … 3C\h, dword aligned).

value 32-bit mask:

- 0 – read-only
- 1 – read/write

See also *"BestConfRegMaskGet" on page 147*

BestConfigScan

Call

```
b_errtype BestConfigScan( b_handletype handle );
```

Description Scans the information stored in the registers of the configuration space header of each PCI device connected to the PCI bus of the system under test.

After this function call you can use *“BestConfigScanPrint” on page 149* to print the scanned information.

CLI Equivalent `BestConfigScan`

CLI Abbreviation `cscan`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

See also –

BestConfigScanPrint

Call

```
b_errtype BestConfigScanPrint(
      b_handletype    handle,
      b_charptrtype   *res_string );
```

Description Returns the information stored in the registers of the configuration space header of each PCI device connected to the PCI bus of the system under test.

Before using this function, call *“BestConfigScan” on page 149* to get the current information from the configuration space headers.

CLI Equivalent `BestConfigScanPrint`

CLI Abbreviation `cscanprnt`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

Output Parameters **res_string** Text string.

See also –

BestPCIConfigCheck

Call `b_errtype BestPCIConfigCheck( b_handletype handle );`

Description Checks the current settings in the configuration space header of the testcard for PCI compliance.

CLI Equivalent `BestPCIConfigCheck`

CLI Abbreviation `cfgcheck`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also –

Expansion ROM Programming Functions

The following functions are used to program the expansion ROM memory space of the testcard:

Function	Result
"BestExpRomByteWrite" on page 152	Writes a byte to the expansion ROM.
"BestExpRomByteRead" on page 151	Reads a byte from the expansion ROM.

How to use the functions is described in *"Expansion ROM Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide.*

BestExpRomByteRead

Call

```
b_errtype BestExpRomByteRead(
      b_handletype   handle,
      b_int32        offset,
      b_int32        *value );
```

Description Reads a byte from the specified offset in the expansion ROM.

CLI Equivalent `BestExpRomByteRead offset=<offset>`

CLI Abbreviation `erbyteread offs=<offset>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

offset Address offset (in byte) in the expansion ROM (0000\h … FFFF\h).

Output Parameters

value Read data byte (bits 0 … 7).

NOTE When a byte is written into the expansion rom and read immediatly after that can return incorrect data. The written data is correct, you cannot directly read from the expansion rom after you have written a byte.

See also *"BestExpRomByteWrite" on page 152*

BestExpRomByteWrite

Call

```
b_errtype BestExpRomByteWrite(
      b_handletype   handle,
      b_int32        offset,
      b_int32        value );
```

Description Writes one byte to the specified offset in the expansion ROM.

CLI Equivalent `BestExpRomByteWrite offset=<offset> value=<value>`

CLI Abbreviation `erbytewrite offs=<offset> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

offset Address offset (in byte) in the expansion ROM (0000\h … FFFF\h).

value Data byte to be written (bits 0 … 7).

NOTE When a byte is written into the expansion ROM and immediately read, it is possible that incorrect data is returned. Bytes cannot be read immediately after data has been written to them.

See also *"BestExpRomByteRead" on page 151*

Data Memory Functions

The following functions access the data memory for master and target transactions:

Function	Result
"BestDataMemInit" on page 153	Initializes the data memory of the testcard by filling it with zeros.
"BestDataMemWrite" on page 154	Writes to the data memory of the testcard.
"BestDataMemRead" on page 153	Reads from the data memory of the testcard.

How to use the functions is described in *“Data Memory and Compare Unit Programming”* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer’s Guide.*

BestDataMemInit

Call

```
b_errtype BestDataMemInit( b_handletype handle );
```

Description Initializes the internal data memory of the testcard and fills it completely with zeros.

CLI Equivalent `BestDataMemInit`

CLI Abbreviation `dataminit`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

See also –

BestDataMemRead

Call

```
b_errtype BestDataMemRead(
      b_handletype   handle,
      b_int32        int_addr,
      b_int32        num_of_bytes,
      b_int8         *data_ptr );
```

Description Reads a data block from the data memory of the testcard to the control PC memory via the control interface.

NOTE Using the CLI restricts the number of bytes to the internal buffer size for the CLI output (8000\h). Within C programs the number of bytes is only restricted by the size of the testcard’s data memory.

CLI Equivalent

```
BestDataMemRead int_addr=<int_addr> num_of_bytes=<num_of_bytes>
[data_ptr>"file path"]
```

CLI Abbreviation

```
datamrd iad=<int_addr> nob=<num_of_bytes> [data>"file path"]
```

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

int_addr Start address of source data within data memory of the testcard. The range depends on the available memory size.

num_of_bytes Number of bytes to be read (maximum of 8000\h when using the CLI).

data_ptr **CLI only.** This parameter allows export of the data to a file. Use a redirection operator, for example: data_ptr>"file path".

Output Parameters **data_ptr** Destination buffer in the memory of the control PC.

See also *"BestDataMemInit" on page 153*
"BestDataMemWrite" on page 154

BestDataMemWrite

Call

```
b_errtype BestDataMemWrite(
      b_handletype   handle,
      b_int32        int_addr,
      b_int32        num_of_bytes,
      b_int8         *data_ptr );
```

Description Writes a data block from the memory of the control PC to the data memory of the testcard. The data memory is shared by master and target. Both can use the data for further testing.

CLI Equivalent `BestDataMemWrite int_addr=<int_addr> num_of_bytes=<num_of_bytes> data_ptr=<{data_list}>`

CLI Abbreviation `datamwr iad=<int_addr> nob=<num_of_bytes> data=<{data_list}>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

int_addr Destination start address for the data within the data memory of the testcard. The range depends on the available memory size.

num_of_bytes Number of bytes to be written.

data_ptr Source data in the system memory of the control PC.

data_ptr **CLI only.** List of data to be transferred when using the CLI (for example, `data={1\h, 2\h, 3\h, 4\h, 5\h, 6\h, 7\h, 8\h}`).

Data may also be imported from a file using a redirection operator (for example, `data_ptr<"file path"`). The input file must contain a sequence of byte values in hexadecimal format, as shown in the following example:

```
01 23 45 67 89 AB CD EF 01 23 45 67 89 AB CD EF
EF CD AB 89 67 45 23 01 EF CD AB 89 67 45 23 01
```

See also *"BestDataMemInit" on page 153*
"BestDataMemRead" on page 153

Host Access Functions

The following functions are used for data transfer to and from the host (control PC):

Function	Result
"BestHostSysMemAccessPrepare" on page 158	Prepares a transfer between the testcard's data memory and the host.
"BestHostSysMemDump64" on page 159	Transfers data from a PCI device to the host system memory.
"BestHostSysMemFill64" page 161	Transfers data from the host system memory to a PCI device.
"BestHostPCIRegSet" on page 157	Sets register of a PCI device.
"BestHostPCIRegGet" on page 156	Reads register of a PCI device.

How to use the functions is described in *"Host Access Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestHostPCIRegGet

Call

```
b_errtype BestHostPCIRegGet(
      b_handletype      handle,
      b_addrspacetype   addrspace,
      b_int32           bus_addr,
      b_int32           *reg_value,
      b_sizetype        size );
```

Description Reads the value from a specific PCI device register in a 32-bit address space—the type of address space determines a Configuration, Memory or I/O Read.

The bus address is a byte address. The function performs single cycle transactions, automatically setting the correct byte enables corresponding to the word size and bus address.

This function only applies to a 32 bit address space.

NOTE *"BestConfRegSet" on page 146* and *"BestConfRegGet" on page 145* access the configuration space registers of the testcard.

CLI Equivalent

```
HostPCIRegGet addrspace=<addrspace> bus_addr=<bus_addr>
wordsize=<size>
```

CLI Abbreviation

```
hprgget space=<addrspace> bad=<bus_addr> size=<size>
```

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

addrspace Type of address space for the register access; see *"b_addrspacetype" on page 227*.

bus_addr PCI bus address.

size Defines the width of the value to be read from the register; see *"b_sizetype" on page 274*.

Output Parameters

reg_value Value read from the register.

See also *"BestHostPCIRegSet" on page 157*

BestHostPCIRegSet

Call

```
b_errtype BestHostPCIRegSet(
      b_handletype      handle,
      b_addrspacetype   addrspace,
      b_int32           bus_addr,
      b_int32           reg_value,
      b_sizetype        size );
```

Description Writes a value to a specific PCI device register—the type of address space determines a Configuration Write, Memory Write or I/O Write.

The bus address is a byte address. The function performs single cycle transactions, automatically setting the correct byte enables corresponding to word size and bus address.

This function only applies to a 32 bit address space.

NOTE *"BestConfRegSet" on page 146* and *"BestConfRegGet" on page 145* access the configuration space registers of the testcard.

CLI Equivalent

```
BestHostPCIRegSet addrspace=<addrspace> bus_addr=<bus_addr>
reg_value=<reg_value> wordsize=<size>
```

CLI Abbreviation

```
hprgset space=<addr_space> bad=<bus_addr> val=<reg_value>
size=<size>
```

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

addrspace Type of address space for the register access; see *"b_addrspacetype" on page 227*.

bus_addr PCI bus address.

reg_value Value to be written to the register.

size Defines the width of the values to be set in the register. See *"b_sizetype" on page 274*.

See also *"BestHostPCIRegGet" on page 156*

BestHostSysMemAccessPrepare

Call

```
b_errtype BestHostSysMemAccessPrepare(
      b_handletype   handle,
      b_int32        buscmd,
      b_int32        bufsize );
```

Description

Prepares the internal address, the command in the master block properties, and a memory buffer for a transfer through the data memory of the testcard. A data verification can be activated.

Call this function before calling *"BestHostSysMemFill64" on page 161* or *"BestHostSysMemDump64" on page 159*.

CLI Equivalent

`BestHostSysMemAccessPrepare buscmd=<buscmd> bufsize=<bufsize>`

CLI Abbreviation

`hsmaprep cmd=<buscmd> buf=<bufsize>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

buscmd Specifies the bus command. See *"b_blkproptype" on page 228*. The command to be specified depends on the access function:

- If using *"BestHostSysMemFill64" on page 161*, the command must be set to B_CMD_MEM_WRITE (Memory Write).
- If using *"BestHostSysMemDump64" on page 159*, the command must be set to B_CMD_MEM_READ (Memory Read).

To enable data verification, OR-combine the bus command with B_HOST_VERIFY. If the verification fails, the access function returns a data compare error or a master abort.

bufsize Specifies the internal memory buffer size in dwords (minimum: 2 dwords).

NOTE

The bufsize must be equal to or larger than the block size specified in *"BestHostSysMemFill64" on page 161* or *"BestHostSysMemDump64" on page 159*.

See also

–

BestHostSysMemDump64

Call

```
b_errtype BestHostSysMemDump64(
      b_handletyp   handle,
      b_int32       bus_addr_low,
      b_int32       bus_addr_high,
      b_int32       num_of_bytes,
      b_int32       blocksize,
      b_int8        *data_ptr );
```

Description Moves data blocks from a device or from the memory in the system under test through the internal data memory of the testcard to the control PC (host). The device is specified by its PCI bus address.

During the master block transfer, the default master attributes are used (master attribute page 0). See *"b_mattrproptype" on page 251*.

NOTE This function changes the settings of generic master properties. See *"b_mastergenproptype" on page 248*.

Each master block transfer can consist of one or more bursts depending on the overall bus traffic and latency timer setting. If a data transfer results in a master abort, the function will *not* return an error, but the master abort is indicated by bit 7 of the status register (block aborted flag). You can read the status register with *"BestStatusRegGet" on page 40*.

Before you call this function for the first time, set the command to **Memory Read** and the size of the buffer in the testcard's internal data memory with *"BestHostSysMemAccessPrepare" on page 158* .

CLI Equivalent

```
BestHostSysMemDump64 bus_addr_low=<bus_addr_low>
bus_addr_high=<bus_addr_high> num_of_bytes=<num_of_bytes>
blocksize=<blocksize> [data>"file path"]
```

CLI Abbreviation

```
hsmdump64 badl=<bus_addr_low> badh=<bus_addr_high>
nob=<num_of_bytes> blk=<blocksize> [data>"file path"]
```

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

bus_addr_low / bus_addr_high PCI bus address from where the data is read. Addresses need not be aligned.

If the high bus address parameter is not equal to zero, the transfers are performed using dual address cycles.

num_of_bytes Number of bytes to be transferred:

- maximum of 128 Kbytes when you use the CLI.
- maximum of 4 Gbytes when you use the C-API.

blocksize Size of the master block transfers in bytes. This blocksize must be smaller or equal to the buffer size specified by *"BestHostSysMemAccessPrepare" on page 158*.

data_ptr Destination in the memory of the control PC.

data **CLI only.** This parameter allows export of the data to a file. Use a redirection operator, for example: `data>"file path"`.

See also *"BestHostSysMemFill64" on page 161*

BestHostSysMemFill64

Call

```
b_errtype BestHostSysMemFill64(
      b_handletype   handle,
      b_int32        bus_addr_low,
      b_int32        bus_addr_high,
      b_int32        num_of_bytes,
      b_int32        blocksize,
      b_int8         *data_ptr );
```

Description Moves data blocks from the control PC (host) through the internal data memory of the testcard to a device or to the memory in the system under test. The device is specified by its PCI bus address.

During the master block transfer, the default master attributes are used (master attribute page 0). See *"b_mattrproptype" on page 251*.

NOTE This function changes the settings of generic master properties. See *"b_mastergenproptype" on page 248*.

Each master block transfer can consist of one or more bursts depending on the overall bus traffic and latency timer setting. If a data transfer results in a master abort, the function will *not* return an error, but the master abort is indicated by bit 7 of the status register (block aborted flag). You can read the status register with *"BestStatusRegGet" on page 40*.

Before you call this function for the first time, set the command to **Memory Write** and the size of the buffer in the testcard's internal data memory with *"BestHostSysMemAccessPrepare" on page 158*.

CLI Equivalent

```
BestHostSysMemFill64 bus_addr_low=<bus_addr_low>
bus_addr_high=<bus_addr_high> num_of_bytes=<num_of_bytes>
blocksize=<blocksize> ( data=<{data_list}> ) | ( data<"file path" )
```

CLI Abbreviation

```
hsmfill64 badl=<bus_addr_low> badh=<bus_addr_high>
nob=<num_of_bytes>
blk=<blocksize> ( data=<{data_list}> ) | ( data<"file path" )
```

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

bus_addr_low / bus_addr_high PCI bus address in PCI memory space to where the data is written. Addresses need not be aligned.

If the high bus address parameter is not equal to zero, the transfers are performed using dual address cycles.

num_of_bytes Number of bytes to be transferred:

- maximum of 128 Kbytes when you use the CLI.
- maximum of 4 Gbytes when you use the C-API.

blocksize Size of the master block transfers in bytes. This blocksize must be smaller or equal to the buffer size specified with *"BestHostSysMemAccessPrepare" on page 158*.

data_ptr Source data in the memory of the control PC.

data **CLI only.** List of data to be transferred when using the CLI (for example, data={1\h, 2\h, 3\h, 4\h, 5\h, 6\h, 7\h, 8\h}.

Data may also be imported from a file using a redirection operator (for example, `data<"file path"`). The input file must contain a sequence of byte values in hexadecimal format, as shown in the following example:

```
01 23 45 67 89 AB CD EF 01 23 45 67 89 AB CD EF
EF CD AB 89 67 45 23 01 EF CD AB 89 67 45 23 01
```

See also *"BestHostSysMemDump64" on page 159*

Interrupt Generation Function

The following function is used to generate a PCI interrupt:

Function	Result
"BestInterruptGenerate" on page 163	Generates a PCI interrupt.

How to use the functions is described in *"Interrupt Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestInterruptGenerate

Call

```
b_errtype BestInterruptGenerate(
      b_handletype   handle,
      b_int32        pci_int );
```

Description Sets the specified PCI interrupt request pin INTA# … INTD#.

To reset the interrupt, clear the corresponding status bit in the status register with *"BestStatusRegClear" on page 39*.

CLI Equivalent `BestInterruptGenerate  pci_int=<pci_int>`

CLI Abbreviation `intgen  int=<pci_int>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

pci_int Interrupts to be generated. See table below.

Multiple interrupts can be set by OR-combining the interrupt values, for example, `pci_int=B_INTA | B_INTC`.

Value (CLI Abbreviation)	Description
B_INTA (inta) … B_INTD (intd)	PCI Interrupt Request A … D (INTA# … INTD#)

See also –

Built-In Test Functions

The following functions program built-in tests:

Function	Result
"BestTestPropDefaultSet" on page 164	Sets the built-in test property to defaults.
"BestTestPropSet" on page 165	Sets a built-in test property.
"BestTestRun" on page 168	Starts a built-in test.
"BestTestResultDump" on page 167	Dumps report and waveform file to the control PC.
"BestTestProtErrDetect" on page 166	Detects protocol errors.

How to use the functions is described in *"Built-In Test Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestTestPropDefaultSet

Call `b_errtype BestTestPropDefaultSet( b_handletype handle );`

Description Sets all the test properties to default values. For a description of properties and default values, see *"b_testproptype" on page 283*.

CLI Equivalent `BestTestPropDefaultSet`

CLI Abbreviation `testprpdefset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestTestPropSet" on page 165*

BestTestPropSet

Call

```
b_errtype  BestTestPropSet(
      b_handletype     handle,
      b_testproptype   testprop,
      b_int32          value );
```

Description Sets a test property. The tests use different properties. Refer to *"Test Commands" on page 169*.

CLI Equivalent `BestTestPropSet  testprop=<testprop> value=<value>`

CLI Abbreviation `testprpset prop=<testprop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

testprop Property to be set; see *"b_testproptype" on page 283*.

value Value to which the property is set, see *"b_testproptype" on page 283*.

See also *"BestTestPropDefaultSet" on page 164*

BestTestProtErrDetect

Call `b_errtype BestTestProtErrDetect( b_handletype handle );`

Description Sets up the analyzer as a PCI protocol error checker. This performs the following:

- Clears the observer status register.
- Clears the “Trace Run Bit” in the Card Status Register.
- Sets the analyzer trigger pattern to trigger on protocol errors.
- Starts the protocol observer and the analyzer.

For further information see:

- *“Protocol Observer Functions” on page 44*
- *“Card Status Functions” on page 39*
- *“Pattern Term Function” on page 64*

CLI Equivalent `BestTestProtErrDetect`

CLI Abbreviation `testpedet`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

See also –

BestTestResultDump

Call

```
b_errtype BestTestResultDump(
      b_handletype    handle,
      b_charptrtype   *filename );
```

Description Saves the test results in a textual report file (`<filename>.rpt`) and in a waveform file (`<filename>.wfm`):

- The *report file* contains the analyzer status, the observer status, and compare errors.
- The *waveform file* contains the trace memory content and can be viewed with the listers of the graphical user interface.

CLI Equivalent `BestTestResultDump file=<filename>`

CLI Abbreviation `testrdump file=<filename>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

filename String containing path and filename (no suffix), for example, `"c:\temp\xx"`.

See also –

BestTestRun

Call

```
b_errtype  BestTestRun(
      b_handletype   handle,
      b_int32        testcmd );
```

Description

Starts the specified test.

Before calling this command, you must set the properties for the specified type of test. The *"Test Commands" on page 169* show which properties the tests use. You can set the properties with *"BestTestPropSet" on page 165*.

Depending on the generic master property B_MGEN_REPEATMODE, the test runs either once or loops infinitely. Refer to *"BestMasterGenPropSet" on page 107*.

You can stop the test with *"BestMasterStop" on page 108*. The test stops automatically after a data compare error (if data comparison is enabled).

CLI Equivalent

```
BestTestRun testcmd=<testcmd>
```

CLI Abbreviation

```
testrun cmd=<testcmd>
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

testcmd Test command, specifying the type of the test to be run. See *"Test Commands" on page 169*.

See also

–

Test Commands

Test Commands (CLI Abbreviation)	Description	Test properties used by the command
B_TSTCMD_TRAFFICMAKE (trafficmake)	Enables the card's master to write data to the card's target over the full PCI bus bandwidth.	B_TST_BANDWIDTH B_TST_BLKLENGTH B_TST_DATAPATTERN B_TST_PROTOCOL
B_TSTCMD_WRITEREAD (writeread)	Performs writes and reads to a PCI device or to system memory.	B_TST_BANDWIDTH B_TST_BLKLENGTH B_TST_DATAPATTERN B_TST_PROTOCOL B_TST_COMPARE B_TST_SOURCEADDR B_TST_NOFBYTES
B_TSTCMD_BLOCKMOVE (blockmove)	Moves a data block from one system memory address to another.	B_TST_BANDWIDTH B_TST_BLKLENGTH B_TST_DATAPATTERN B_TST_PROTOCOL B_TST_COMPARE B_TST_SOURCEADDR B_TST_DESTINADDR B_TST_NOFBYTES
B_TSTCMD_READ (read)	Performs read transactions from a PCI device or from system memory.	B_TST_BANDWIDTH B_TST_BLKLENGTH B_TST_PROTOCOL B_TST_SOURCEADDR B_TST_NOFBYTES

Hot Swap Debugging Functions

The following functions are used for hot swap debugging:

Function	Result
"BestCPCIStatusSet" on page 170	Sets a bit in the hot swap control register.
"BestCPCIStatusGet" on page 170	Reads a bit from the hot swap control register.
"BestCPCIPinSet" on page 171	Sets pin used for hot swapping.
"BestCPCIPinGet" on page 171	Reads pin used for hot swapping.

How to use the functions is described in *"Hot Swap Test Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestCPCIStatusGet

Call

```
b_errtype  BestCPCIStatusGet(
      b_handletype       handle,
      b_cpcistatustype   status,
      b_int32            *value );
```

Description Reads a bit from the hot swap control register (HS_CSR). For details, refer to the CompactPCI Hot Swap Specification.

CLI equivalent `BestCPCIStatusGet status=<status>`

CLI abbreviation `cpcisget stat=<status>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

status Property to be read; see *"b_cpcistatustype" on page 233*.

Output Parameters

value Value of the property; see *"b_cpcistatustype" on page 233*.

See also *"BestCPCIStatusSet" on page 170*

BestCPCIStatusSet

Call

```
b_errtype  BestCPCIStatusSet(
      b_handletype       handle,
      b_cpcistatustype   status,
      b_int32            value );
```

Description Sets a bit in the hot swap control register (HS_CSR). For details, refer to the CompactPCI Hot Swap Specification.

CLI Equivalent `BestCPCIStatusSet status=<status> value=<value>`

CLI Abbreviation `cpcisset stat=<testprop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

status Property to be set; see *"b_cpcistatustype" on page 233*.

value Value of the property; see *"b_cpcistatustype" on page 233*.

See also *"BestCPCIStatusGet" on page 170*

BestCPCIPinGet

Call

```
b_errtype  BestCPCIPinGet(
      b_handletype    handle,
      b_cpcipintype   pin,
      b_int32         *value );
```

Description Reads pin used for control of hot swapping (CompactPCI). For details, refer to the CompactPCI Hot Swap Specification.

CLI Equivalent `BestCPCIPinGet pin=<pin>`

CLI Abbreviation `cpcipget pin=<pin>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

pin Pin to be read; see *"b_cpcipintype" on page 232*.

Output Parameters

value Value of the pin; *"b_cpcipintype" on page 232*.

See also *"BestCPCIPinSet" on page 171*

BestCPCIPinSet

Call

```
b_errtype  BestCPCIPinSet(
      b_handletype    handle,
      b_cpcipintype   pin,
      b_int32         value );
```

Description Sets pin used for hot swapping (CompactPCI). For details, refer to CompactPCI Hot Swap Specification.

CLI Equivalent `BestCPCIPinSet pin=<pin> value=<value>`

CLI Abbreviation `cpcipset pin=<pin> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

pin Pin to be set; see *"b_cpcipintype" on page 232*.

value Value of the pin; see *"b_cpcipintype" on page 232*.

See also *"BestCPCIPinGet" on page 171*

Interface Control Functions

The PCI Interface Control functions are divided into the following sections:

- *"Trigger I/O Sequencer Programming Functions" on page 174*
- *"Display Functions" on page 181*
- *"Mailbox Functions" on page 182*

Trigger I/O Sequencer Programming Functions

The following functions are used to program the external trigger I/O lines:

Function	Result
"BestTrigIOGenPropDefaultSet" on page 175	Sets all generic trigger I/O sequencer properties to default values.
"BestTrigIOGenPropGet" on page 175	Reads the value of a generic trigger I/O sequencer property.
"BestTrigIOGenPropSet" on page 176	Sets the value of a generic trigger I/O sequencer property.
"BestTrigIOSeqPropDefaultSet" on page 177	Sets all properties in the trigger I/O sequencer description table to default values.
"BestTrigIOSeqTranPropDefaultSet" on page 179	Sets all properties of a transient in the trigger I/O sequencer description table to default values.
"BestTrigIOSeqTranPropSet" on page 180	Sets numeric transition properties ("state" or "next state").
"BestTrigIOSeqTranCondPropSet" on page 178	Sets conditions in the trigger I/O sequencer description table.
"BestTrigIOSeqProg" on page 176	Writes the sequencer description table to sequencer memory.
"BestTrigIORun" on page 177	Starts the trigger I/O sequencer.
"BestTrigIOStop" on page 180	Stops the trigger I/O sequencer.

How to use the functions is described in *"Trigger I/O Sequencer Programming"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide.*

BestTrigIOGenPropDefaultSet

Call `b_errtype BestTrigIOGenPropDefaultSet( b_handletype handle );`

Description Sets all generic properties of the trigger I/O sequencer to default values.

CLI Equivalent `BestTrigIOGenPropDefaultSet`

CLI Abbreviation `tiogenprpdefset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"b_trigioseqgenproptype" on page 287*
"BestTrigIOGenPropGet" on page 175
"BestTrigIOGenPropSet" on page 176

BestTrigIOGenPropGet

Call

```
b_errtype BestTrigIOGenPropGet(
      b_handletype          handle,
      b_trigiogenproptype   trigiogenprop,
      b_int32               *value );
```

Description Queries generic properties of the trigger I/O sequencer. Generic properties determine the preload value of the feedback counter and the trigger I/O output settings.

CLI Equivalent `BestTrigIOGenPropGet trigiogenprop=<trigiogenprop>`

CLI Abbreviation `tiogenprpget prop=<trigiogenprop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

trigiogenprop Property to be queried; see *"b_trigioseqgenproptype" on page 287*.

Output Parameters **value** Value of the queried property; see *"b_trigioseqgenproptype" on page 287*.

See also *"BestTrigIOGenPropDefaultSet" on page 175*
"BestTrigIOGenPropSet" on page 176

BestTrigIOGenPropSet

Call

```
b_errtype BestTrigIOGenPropSet(
      b_handletype          handle,
      b_trigiogenproptype   trigiogenprop,
      b_int32               value );
```

Description Sets generic properties of the trigger I/O sequencer. Generic properties determine the preload value of the feedback counter and the trigger I/O output settings.

CLI Equivalent `BestTrigIOGenPropSet trigiogenprop=<trigiogenprop> value=<value>`

CLI Abbreviation `tiogenprpset prop=<trigiogenprop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

trigiogenprop Property to be set; see *"b_trigioseqgenproptype" on page 287*.

value Value to which the property is set; see *"b_trigioseqgenproptype" on page 287*.

See also *"BestTrigIOGenPropDefaultSet" on page 175*
"BestTrigIOGenPropGet" on page 175

BestTrigIOSeqProg

Call `b_errtype BestTrigIOSeqProg( b_handletype handle );`

Description Writes the information stored in the trigger I/O sequencer description table to the sequencer memory.

This function also checks whether transition conditions of one state are consistent. If they are not, the function returns an error.

To complete sequencer programming, the pattern terms must be programmed with *"BestPattSet" on page 64*.

CLI Equivalent `BestTrigIOSeqProg`

CLI Abbreviation `tiosprog`

Return Value Error code; see *"b_errtype" on page 241*.

Input Paremeters **handle** Handle to identify the session.

See also –

BestTrigIORun

Call `b_errtype BestTrigIORun( b_handletype handle );`

Description Starts the trigger I/O sequencer (and the performance counters).

CLI Equivalent `BestTrigIORun`

CLI Abbreviation `tiorun`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestPerfRun" on page 91*
"BestTrigIOStop" on page 180

BestTrigIOSeqPropDefaultSet

Call `b_errtype BestTrigIOSeqPropDefaultSet( b_handletype handle );`

Description Initializes the trigger I/O sequencer description table and sets all properties to default values.

For a description of properties and default values, refer to *"b_trigioseqtranproptype" on page 289* and *"b_trigseqtrancondproptype" on page 290*.

CLI Equivalent `BestTrigIOSeqPropDefaultSet`

CLI Abbreviation `tiosprpdefset`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also –

BestTrigIOSeqTranCondPropSet

Call

```
b_errtype BestTrigIOSeqTranCondPropSet(
      b_handletype                 handle,
      b_int32                      transient,
      b_trigioseqtrancondproptype  trigioseqtrancondprop,
      b_charptrtype                condition );
```

Description

Sets condition properties in the trigger I/O sequencer description table.

Condition properties can be the transition condition, trigger condition, storage qualifier condition, or conditions to decrement or preload the feedback counter.

CLI Equivalent

```
BestTrigIOSeqTranCondPropSet transient=<transient>
trigioseqtrancondprop=<trigioseqtrancondprop> condition=<condition>
```

CLI Abbreviation

```
tiostrancprpset tran=<transient> prop=<trigioseqtrancondprop>
con=<condition>
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

transient Number of the transient (0 … 251).

trigioseqtrancondprop Property to be set; see *"b_trigioseqtrancondproptype" on page 288*.

condition Condition string to which the property is set. The string must be written in quotation marks. See *"Conditions Reference" on page 74*.

See also

"BestTrigIOSeqTranPropDefaultSet" on page 179
"BestTrigIOSeqTranPropSet" on page 180

BestTrigIOSeqTranPropDefaultSet

Call

```
b_errtype BestTrigIOSeqTranPropDefaultSet(
      b_handletype   handle,
      b_int32        transient );
```

Description

Sets all properties of a transient in the trigger I/O sequencer description table to default values.

For a description of properties and default values, refer to *"b_trigioseqtrancondproptype" on page 288* and *"b_trigioseqtranproptype" on page 289*.

CLI Equivalent

`BestTrigIOSeqTranPropDefaultSet transient=<transient>`

CLI Abbreviation

`tiostranprpdefset tran=<transient>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

transient Transient number (0 … 251).

See also

"BestTrigIOSeqTranCondPropSet" on page 178
"BestTrigIOSeqTranPropSet" on page 180

BestTrigIOSeqTranPropSet

Call

```
b_errtype BestTrigIOSeqTranPropSet(
      b_handletype             handle,
      b_int32                  transient,
      b_trigioseqtranproptype  trigioseqtranprop,
      b_int32                  value );
```

Description Sets a numeric transition property (“state” or “next state”) in the trigger I/O sequencer description table.

CLI Equivalent

```
BestTrigIOSeqTranPropSet transient=<transient>
trigioseqtranprop=<trigioseqtranprop> value=<value>
```

CLI Abbreviation

```
tiostranprpset tran=<transient> prop=<trigioseqtranprop>
val=<value>
```

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

transient Number of the transient (0 … 251).

trigioseqtranprop Property to be set; see *“b_trigioseqtranproptype” on page 289*.

value Value to which the property is set; see *“b_trigioseqtranproptype” on page 289*.

See also

“BestTrigIOSeqTranPropDefaultSet” on page 179
“BestTrigIOSeqTranCondPropSet” on page 178

BestTrigIOStop

Call

```
b_errtype BestTrigIOStop( b_handletype handle );
```

Description Stops the trigger I/O sequencer (and the performance counters).

CLI Equivalent

```
BestTrigIOStop
```

CLI Abbreviation

```
tiostop
```

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

See also

“BestTrigIORun” on page 177
“BestPerfStop” on page 97

Display Functions

The following functions are used to control the LED display:

Function	Result
"BestDisplayPropSet" on page 181	Sets the display mode.
"BestDisplayWrite" on page 182	Writes a value to the display.

How to use the functions is described in *"LED Controlling and Display Functions Overview"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestDisplayPropSet

Call

```
b_errtype BestDisplayPropSet(
      b_handletype   handle,
      b_int32        value );
```

Description Sets the mode of the LED display. Use this function to select "user mode" before writing values to the display.

CLI Equivalent `BestDisplayPropSet value=<value>`

CLI Abbreviation `dprpset val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

value The display provides the following modes:

Display Mode (CLI abbreviation)	Description
B_DISP_USER (user)	User mode. Select this mode to write to the display.
default: B_DISP_CARD (card)	The display shows internal system states.

See also *"BestDisplayWrite" on page 182*

BestDisplayWrite

Call

```
b_errtype BestDisplayWrite(
      b_handletype   handle,
      b_int32        value );
```

Description Writes a value to the LED display.

Before using this function, ensure that “user mode” is selected for the display. You can query the selected mode with *“BestDisplayPropSet” on page 181*.

CLI Equivalent `BestDisplayWrite value=<value>`

CLI Abbreviation `dwrite val=<value>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

value 5-bit value to be displayed.

See also –

Mailbox Functions

The following functions are used for communication between testcard and test environment via the PCI interface:

Function	Result
“BestMailboxReceiveRegRead” on page 183	Reads data from mailbox to control PC.
“BestMailboxSendRegWrite” on page 184	Sends data from control PC to mailbox.
“BestPCICfgMailboxReceiveRegRead” on page 185	Reads data from mailbox via PCI.
“BestPCICfgMailboxSendRegWrite” on page 186	Sends data to mailbox via PCI.

How to use the functions is described in *“Mailbox Programming”* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer’s Guide*.

BestMailboxReceiveRegRead

Call

```
b_errtype BestMailboxReceiveRegRead(
      b_handletype   handle,
      b_int32        *value,
      b_int32        *status );
```

Description This function is used on the control PC to receive a value from the mailbox via the control interface. The function returns the mailbox value and a status flag.

CLI Equivalent `BestMailboxReceiveRegRead`

CLI Abbreviation `mrregread`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

Output Parameters **value** Value read from the mailbox.

status Status flag:

- 0 – The mailbox was empty. The transfer failed.
- 1 – The transfer was successful.

See also *“BestMailboxSendRegWrite” on page 184*
“BestPCICfgMailboxReceiveRegRead” on page 185

BestMailboxSendRegWrite

Call

```
b_errtype BestMailboxSendRegWrite(
      b_handletype   handle,
      b_int32        value,
      b_int32        *status );
```

Description

This function is used on the control PC to send a value to the mailbox via the control interface.

If the mailbox is occupied (by unread data, see status flag below), the function will not write the value to the mailbox.

CLI Equivalent

`BestMailboxSendRegWrite value=<value>`

CLI Abbreviation

`msregwrite val=<value>`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

value Value to send to the mailbox.

Output Parameters

status Status flag:

- 0 – Unread data was found in the mailbox. The transfer failed.
- 1 – The transfer was successful.

See also

"BestMailboxReceiveRegRead" on page 183
"BestPCICfgMailboxSendRegWrite" on page 186

BestPCICfgMailboxReceiveRegRead

Call

```
b_errtype BestPCICfgMailboxReceiveRegRead (
      b_int32   devid,
      b_int32   *value,
      b_int32   *status );
```

Description

This function is used on the system under test to read a value from the mailbox via the PCI bus. The function returns the mailbox value and the status flag.

To identify the testcard within the PCI system, first call *"BestDevIdentifierGet" on page 19*. The device identifier returned by this function can be used directly.

NOTE

When checking for errors, use *non-handle-based error checking*. See *"Error Handling" on page 223*.

CLI Equivalent

No CLI equivalent.

CLI Abbreviation

No CLI abbreviation.

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

devid Device identifier of the testcard as returned by *"BestDevIdentifierGet" on page 19*.

Output Parameters

value Value read from the mailbox.

status Status flag:

- 0 – The mailbox was empty. The transfer failed.
- 1 – The transfer was successful.

See also

"BestMailboxReceiveRegRead" on page 183
"BestPCICfgMailboxSendRegWrite" on page 186

BestPCICfgMailboxSendRegWrite

Call

```
b_errtype BestPCICfgMailboxSendRegWrite(
      b_int32   devid,
      b_int32   value,
      b_int32   *status );
```

Description

This function is used on the system under test to write a value to the mailbox via the PCI bus.

To identify the testcard within the PCI system, first call *"BestDevIdentifierGet" on page 19*. The device identifier returned by this function can be used directly.

If the mailbox is occupied (by unread data), the function will not write the value to the mailbox.

NOTE

When checking for errors, use *non-handle-based error checking*. See *"Error Handling" on page 223*.

CLI Equivalent

No CLI equivalent.

CLI Abbreviation

No CLI abbreviation.

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

devid Device identifier of the testcard as returned by *"BestDevIdentifierGet" on page 19*.

value Value to send to the mailbox.

Output Parameters

status Status flag:

- 0 – Unread data was found in the mailbox. The transfer failed.
- 1 – The transfer was successful.

See also

"BestPCICfgMailboxReceiveRegRead" on page 185
"BestMailboxSendRegWrite" on page 184

Protocol Permutator and Randomizer Functions

The PCI Protocol Permutator and Randomizer (PPR) functions can be divided into several categories:

- *“PPR Administrating Functions” on page 187*
- *“Master Block Permutation Functions” on page 192*
- *“Master Attribute Permutation Functions” on page 200*
- *“PPR Report Functions” on page 208*
- *“Target Attribute Permutation Functions” on page 214*

PPR Administrating Functions

The following functions are used to initialize and deinitialize the PCI Protocol Permutator and Randomizer software:

Function	Result
“BestPprInit” on page 191	Initializes the PPR software.
“BestPprDelete” on page 188	Frees all memory allocated by the software.
“BestPprGenPropDefaultSet” on page 188	Sets all PPR generic properties to default values.
“BestPprGenPropSet” on page 190	Sets values of PPR generic properties.
“BestPprGenPropGet” on page 189	Reads values of PPR generic properties.

How to use the functions is described in *“PPR Administration”* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer’s Guide*.

BestPprDelete

Call `b_errtype BestPprDelete( b_handletype handle );`

Description Frees all the memory space allocated by the PCI PPR software. You must use this function when you finish working with the software.

CLI Equivalent `BestPprDelete`

CLI Abbreviation `pprd`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestPprInit" on page 191*

BestPprGenPropDefaultSet

Call `b_errtype BestPprGenPropDefaultSet( b_handletype handle );`

Description Sets all generic properties of the PCI PPR software to default values. For a description of properties and default values, see *"bppr_genproptype" on page 300*.

CLI Equivalent `BestPprGenPropDefaultSet`

CLI Abbreviation `pprgpds`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestPprGenPropSet" on page 190*
"BestPprGenPropGet" on page 189

BestPprGenPropGet

Call

```
b_errtype BestPprGenPropGet(
      b_handletype       handle,
      bppr_genproptype   prop,
      b_int32            *value );
```

Description Reads generic property values of the PCI PPR software.

The generic properties are PCI bus speed and width, expected number of clocks per data transfer and random seed.

CLI Equivalent `BestPprGenPropGet prop=<prop>`

CLI Abbreviation `pprgpg p=<prop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

prop Property to be read; see *"bppr_genproptype" on page 300*.

Output Parameters

value Value of the read property; see *"bppr_genproptype" on page 300*.

See also

"BestPprGenPropSet" on page 190
"BestPprGenPropDefaultSet" on page 188

BestPprGenPropSet

Call

```
b_errtype BestPprGenPropSet(
      b_handletype       handle,
      bppr_genproptype   prop,
      b_int32            value );
```

Description

Sets generic property values of the PCI PPR software.

The generic properties are PCI bus speed and width, expected number of clocks per data transfer and random seed.

CLI Equivalent

`BestPprGenPropSet prop=<prop> value=<value>`

CLI Abbreviation

`pprgps p=<prop> val=<value>`

Return Value

Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

prop Property to be set; see *“bppr_genproptype” on page 300*.

value Value to which the property is set; see *“bppr_genproptype” on page 300*.

See also

“BestPprGenPropGet” on page 189
“BestPprGenPropDefaultSet” on page 188

BestPprInit

Call

`b_errtype BestPprInit( b_handletype handle );`

Description

Initializes the PCI PPR software and sets all properties of this software to default values. This function must be called before any other PCI PPR function.

At the end of the test you must call *"BestPprDelete" on page 188* to free the memory.

If you just want to set the generic properties to default values, see *"BestPprGenPropDefaultSet" on page 188*.

If you just want to set specific parts of the program to default values, see *"BestPprBlockInit" on page 194*, *"BestPprMAttrInit" on page 202*, *"BestPprTAttrInit" on page 216*, or *"BestPprReportPropDefaultSet" on page 209*.

CLI Equivalent

`BestPprInit`

CLI Abbreviation

`ppri`

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

NOTE

If you call BestPprInit twice with the same session handle, the memory allocated by the first call is freed automatically.

See also

"bppr_genproptype" on page 300

Master Block Permutation Functions

The following functions are used to prepare and to perform a master block permutation:

Function	Result
"BestPprBlockInit" on page 194	Initializes the block permutator.
"BestPprBlockPermPropDefaultSet" on page 195	Sets all block permutation properties to default values.
"BestPprBlockPermPropSet" on page 196	Sets values of block permutation properties.
"BestPprBlockPermPropGet" on page 195	Reads values of block permutation properties.
"BestPprBlockVariationDefaultSet" on page 197	Sets all block variation parameters to default values.
"BestPprBlockVariationSet" on page 199	Sets the value list and the corresponding algorithm of a block variation parameter.
"BestPprBlockVariationGet" on page 198	Reads the value list and the corresponding algorithm of a block variation parameter.
"BestPprBlockGenerate" on page 194	Generates a compound block.
"BestPprBlockResultGet" on page 197	Reads a result parameter.
"BestPprBlockCoverageGet" on page 193	Returns coverage and repetition length.

How to use the functions is described in *"Programming Master Block Permutations"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestPprBlockCoverageGet

Call

```
b_errtype BestPprBlockCoverageGet(
      b_handletype   handle,
      b_int32        nmb_param,
      b_int32        *param_list,
      b_int32        *rep_len,
      b_int32        *covered );
```

Description Internally performs the permutation algorithm with the parameters currently set. “Internally” means nothing is downloaded to the testcard. This function determines whether all permutations of a given tuple of block variation parameters are covered, and returns the repetition length of the tuple.

CLI Equivalent No CLI equivalent. To get coverage information with the CLI, use *“BestPprReportWrite” on page 212*.

CLI Abbreviation No CLI abbreviation.

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

nmb_param Number of block variation parameters in the tuple.

param_list Array that defines the tuple of which the coverage is to be tested. The tuple holds nmb_param entries. The entries must be valid values of *“bppr_blkvarparamtype” on page 297*.

Output Parameters

rep_len Returns the repetition length of the tuple (that is the number of permutations of the tuple).

covered Returned values:

- 1 – if the tuple is covered after complete execution of the master block page created by the permutation algorithm.
- 0 – if this is not the case.

See also –

BestPprBlockGenerate

Call `b_errtype BestPprBlockGenerate( b_handletype handle );`

Description Generates a compound block using the properties shown in the table below and downloads this compound block to the testcard.

A compound block consists of blocks with differing block variation parameters. The number of these blocks depends on the master block page size (MBPS).

Properties of Master Block Page Generation	Adjusted by Function
Generic setup properties	*"BestPprGenPropSet" on page 190*
Master block properties	*"BestMasterBlockPropSet" on page 125*
Block permutation properties	*"BestPprBlockPermPropSet" on page 196*
Block variation parameters	*"BestPprBlockVariationSet" on page 199*

CLI Equivalent `BestPprBlockGenerate`

CLI Abbreviation `pprbg`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"bppr_blkvarparamtype" on page 297*

BestPprBlockInit

Call `b_errtype BestPprBlockInit( b_handletype handle );`

Description Initializes the block permutator and sets all block permutation properties to default values.

To set individual properties to default values, use either the function *"BestPprBlockPermPropDefaultSet" on page 195* or *"BestPprBlockVariationDefaultSet" on page 197*.

CLI Equivalent `BestPprBlockInit`

CLI Abbreviation `pprbi`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also –

BestPprBlockPermPropDefaultSet

Call

```
b_errtype BestPprBlockPermPropDefaultSet( b_handletype handle );
```

Description

Sets the block permutation properties to default values. For a description of properties and default values, see *"bppr_blkpermproptype" on page 294*.

CLI Equivalent

```
BestPprBlockPermPropDefaultSet
```

CLI Abbreviation

```
pprbppds
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

See also

"BestPprBlockPermPropSet" on page 196
"BestPprBlockPermPropGet" on page 195

BestPprBlockPermPropGet

Call

```
b_errtype BestPprBlockPermPropGet(
      b_handletype           handle,
      bppr_blkpermproptype   prop,
      b_int32                *value );
```

Description

Reads the value of a block permutation property.

CLI Equivalent

```
BestPprBlockPermPropGet prop=<prop>
```

CLI Abbreviation

```
pprbppg p=<prop>
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

prop Property to be read; see *"bppr_blkpermproptype" on page 294*.

Output Parameters

value Value of the read property; see *"bppr_blkpermproptype" on page 294*.

See also

"BestPprBlockPermPropDefaultSet" on page 195
"BestPprBlockPermPropSet" on page 196

BestPprBlockPermPropSet

Call

```
b_errtype BestPprBlockPermPropSet(
      b_handletype          handle,
      bppr_blkpermproptype  prop,
      b_int32               value );
```

Description Sets a block permutation property.

CLI Equivalent `BestPprBlockPermPropSet prop=<prop> value=<value>`

CLI Abbreviation `pprbpps p=<prop> val=<value>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

prop Property to be set; see *“bppr_blkpermproptype” on page 294*.

value Value to which the property is set; see *“bppr_blkpermproptype” on page 294*.

See also

“BestPprBlockPermPropDefaultSet” on page 195
“BestPprBlockPermPropGet” on page 195

BestPprBlockResultGet

Call

```
b_errtype BestPprBlockResultGet(
      b_handletype              handle,
      bppr_blkresultparamtype   param,
      b_int32                   *value );
```

Description Internally performs the permutation algorithm with the parameters currently set. “Internally” means nothing is downloaded to the testcard. This function reads the value of a selected master block result parameter.

CLI Equivalent `BestPprBlockResultGet param=<param>`

CLI Abbreviation `pprbrg p=<param>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

param Result parameter to be read; see *“bppr_blkresultparamtype” on page 296*.

Output Parameters

value Value of the read result parameter.

See also –

BestPprBlockVariationDefaultSet

Call

```
b_errtype BestPprBlockVariationDefaultSet( b_handletype handle );
```

Description Sets the block variation parameters to default values. For a description of properties and default values, see *“bppr_blkvarparamtype” on page 297*.

CLI Equivalent `BestPprBlockVariationDefaultSet`

CLI Abbreviation `pprbvds`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

See also

“BestPprBlockVariationGet” on page 198
“BestPprBlockVariationSet” on page 199

BestPprBlockVariationGet

Call

```
b_errtype BestPprBlockVariationGet(
      b_handletype          handle,
      bppr_blkvarparamtype  param,
      b_charptrtype         *value_list,
      bppr_algorithmtype    *algorithm );
```

Description

Reads the value list and the corresponding algorithm of a block variation parameter.

The block variation parameters are alignments, block sizes, byte enables or PCI bus commands.

CLI Equivalent

`BestPprBlockVariationGet param=<param>`

CLI Abbreviation

`pprbvg p=<param>`

Return Value

Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

param Parameter to be read; see *“bppr_blkvarparamtype” on page 297*.

Output Parameters

value_list List of permutated values. For the syntax of the list, see *“Value Lists” on page 298*.

If the PCI PPR software has not yet been initialized with *“BestPprInit” on page 191*, the list will be empty and an error will be returned.

algorithm Returns the algorithm used to pick values from the value list. See *“bppr_algorithmtype” on page 292*.

See also

“BestPprBlockVariationDefaultSet” on page 197
“BestPprBlockVariationSet” on page 199

BestPprBlockVariationSet

Call

```
b_errtype BestPprBlockVariationSet(
      b_handletype          handle,
      bppr_blkvarparamtype  param,
      b_charptrtype         value_list,
      bppr_algorithmtype    algorithm );
```

Description

Defines the value list for a block variation parameter and selects the algorithm used to pick values from this list.

Block variation parameters are alignments, block sizes, byte enables or PCI bus commands. To select PCI bus commands, specific algorithms are available. See *"bppr_algorithmtype: B_BLK_CMDS Details" on page 292*.

CLI Equivalent

```
BestPprBlockVariationSet param=<param> value_list=<value_list>
algorithm=<algorithm>
```

CLI Abbreviation

```
pprbvs p=<param> list=<value_list> alg=<algorithm>
```

Return Value

Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

param Block variation parameter; see *"bppr_blkvarparamtype" on page 297*.

value_list List of values to be permutated according to the selected algorithm. For the syntax of the list, see *"Value Lists" on page 298*.

algorithm Algorithm used to pick values from the value list; see *"bppr_algorithmtype" on page 292*.

See also

"BestPprBlockVariationDefaultSet" on page 197
"BestPprBlockVariationGet" on page 198

Master Attribute Permutation Functions

The following functions are used to prepare and to perform the permutation of the master attributes:

Function	Result
"BestPprMAttrInit" on page 202	Initializes the master attribute permutator.
"BestPprMAttrPermPropDefaultSet" on page 203	Sets all master attribute properties to default values.
"BestPprMAttrPermPropSet" on page 204	Sets values of master attribute permutation properties.
"BestPprMAttrPermPropGet" on page 203	Reads values of master attribute permutation properties.
"BestPprMAttrVariationDefaultSet" on page 205	Sets all master attribute variation parameters to default values.
"BestPprMAttrVariationSet" on page 207	Sets the value list and the corresponding algorithm of a master attribute.
"BestPprMAttrVariationGet" on page 206	Reads the value list and the corresponding algorithm of a master attribute.
"BestPprMAttrGenerate" on page 202	Generates a master attribute page.
"BestPprMAttrResultGet" on page 205	Reads a result parameter.
"BestPprMAttrCoverageGet" on page 201	Returns coverage and repetition length.

How to use the functions is described in *"Programming Master Attribute Permutations"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestPprMAttrCoverageGet

Call

```
b_errtype BestPprMAttrCoverageGet(
      b_handletype      handle,
      b_int32           nmb_attr,
      b_mattrproptype   *attr_list,
      b_int32           *rep_len,
      b_int32           *covered );
```

Description Internally performs the permutation algorithm with the parameters currently set. "Internally" means that nothing is downloaded to the testcard. This function determines whether all permutations of a given tuple of master attributes are covered, and returns the repetition length of the tuple.

CLI Equivalent No CLI equivalent. To get coverage information with the CLI, use *"BestPprReportWrite" on page 212*.

CLI Abbreviation No CLI abbreviation.

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

nmb_attr Number of master attributes in the tuple.

attr_list This array defines the tuple with the coverage to be tested. The tuple holds nmb_attr entries. The entries must be valid values of *"b_mattrproptype" on page 251*.

Output Parameters **rep_len** Returns the repetition length of the tuple (that is the number of permutations of the tuple).

covered Returned values:

- 1 – if the tuple is covered after complete execution of the master attribute page created by the permutation algorithm.
- 0 – if this is not the case.

See also –

BestPprMAttrGenerate

Call `b_errtype BestPprMAttrGenerate( b_handletype handle );`

Description Generates a master attribute page using the properties shown in the table below and downloads this master attribute page to the testcard.

Properties of Master Attribute Page Generation	Adjusted by Function
Generic setup properties	*"BestPprGenPropSet" on page 190*
Master attribute permutation properties	*"BestPprMAttrPermPropSet" on page 204*
Master attribute variation parameters	*"BestPprMAttrVariationSet" on page 207*

CLI Equivalent `BestPprMAttrGenerate`

CLI Abbreviation `pprmag`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"b_mattrproptype" on page 251*

BestPprMAttrInit

Call `b_errtype BestPprMAttrInit( b_handletype handle );`

Description Initializes the master attribute permutator and sets all of its properties to default values.

To set individual properties to default values, use either the function *"BestPprMAttrPermPropDefaultSet" on page 203* or *"BestPprMAttrVariationDefaultSet" on page 205*.

CLI Equivalent `BestPprMAttrInit`

CLI Abbreviation `pprmai`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also –

BestPprMAttrPermPropDefaultSet

Call

```
b_errtype BestPprMAttrPermPropDefaultSet( b_handletype handle );
```

Description

Sets the master attribute permutation properties to default values. For a description of properties and default values, see *“bppr_mattrpermproptype” on page 301*.

CLI Equivalent

```
BestPprMAttrPermPropDefaultSet
```

CLI Abbreviation

```
pprmappds
```

Return Value

Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

See also

“BestPprMAttrPermPropGet” on page 203
“BestPprMAttrPermPropSet” on page 204

BestPprMAttrPermPropGet

Call

```
b_errtype BestPprMAttrPermPropGet(
      b_handletype             handle,
      bppr_mattrpermproptype   prop,
      b_int32                  *value );
```

Description

Reads a master attribute permutation property.

CLI Equivalent

```
BestPprMAttrPermPropGet prop=<prop>
```

CLI Abbreviation

```
pprmappg p=<prop>
```

Return Value

Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

prop Property to be read; see *“bppr_mattrpermproptype” on page 301*.

Output Parameters

value Value of the read property; see *“bppr_mattrpermproptype” on page 301*.

See also

“BestPprMAttrPermPropDefaultSet” on page 203
“BestPprMAttrPermPropSet” on page 204

BestPprMAttrPermPropSet

Call

```
b_errtype BestPprMAttrPermPropSet(
      b_handletype             handle,
      bppr_mattrpermproptype   prop,
      b_int32                  value );
```

Description Sets a master attribute permutation property.

CLI Equivalent `BestPprMAttrPermPropSet prop=<prop> value=<value>`

CLI Abbreviation `pprmapps p=<prop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

prop Property to be set; see *"bppr_mattrpermproptype" on page 301*.

value Value to which the property is set; see *"bppr_mattrpermproptype" on page 301*.

See also *"BestPprMAttrPermPropDefaultSet" on page 203*
"BestPprMAttrPermPropGet" on page 203

BestPprMAttrResultGet

Call

```
b_errtype BestPprMAttrResultGet(
      b_handletype                handle,
      bppr_mattrresultparamtype   param,
      b_int32                     *value );
```

Description Internally performs the permutation algorithm with the parameters currently set. “Internally” means nothing is downloaded to the testcard. This function reads the value of a selected master attribute result parameter.

CLI Equivalent `BestPprMAttrResultGet param=<param>`

CLI Abbreviation `pprmarg p=<param>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

param Result parameter to be read; see *“bppr_mattrresultparamtype” on page 302*.

Output Parameters **value** Value of the read result parameter; see *“bppr_mattrresultparamtype” on page 302*.

See also –

BestPprMAttrVariationDefaultSet

Call

```
b_errtype BestPprMAttrVariationDefaultSet( b_handletype handle );
```

Description Sets the master attribute variation parameters to default values. For a description of parameters and default values, see *“b_mattrproptype” on page 251*.

CLI Equivalent `BestPprMAttrVariationDefaultSet`

CLI Abbreviation `pprmavds`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *“BestPprMAttrVariationGet” on page 206*
“BestPprMAttrVariationSet” on page 207

BestPprMAttrVariationGet

Call

```
b_errtype BestPprMAttrVariationGet(
      b_handletype         handle,
      bppr_mattrproptype   attribute,
      b_charptrtype        *value_list,
      bppr_algorithmtype   *algorithm );
```

Description Reads the variation properties of a master attribute.

CLI Equivalent `BestPprMAttrVariationGet attribute=<attribute>`

CLI Abbreviation `pprmvg attr=<attribute>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

attribute Master attribute to be read; see *"b_mattrproptype" on page 251*.

For more information on master attributes, see *Agilent E2940A Exerciser User's Guide*.

Output Parameters

value_list Returns a pointer to a string containing a comma-separated list of values. The values in this list are permutated according to the selected algorithm. For a description of the string's contents, refer to *"Numeric values" on page 299*.

If the PCI PPR software has not yet been initialized with *"BestPprBlockInit" on page 194* or *"BestPprInit" on page 191*, the list will be empty and an error will be returned.

algorithm Returns the algorithm used to pick values from the value list; see *"bppr_algorithmtype" on page 292*.

See also

"BestPprMAttrVariationDefaultSet" on page 205
"BestPprMAttrVariationSet" on page 207

BestPprMAttrVariationSet

Call

```
b_errtype BestPprMAttrVariationSet(
      b_handletype         handle,
      b_mattrproptype      attribute,
      b_charptrtype        value_list,
      bppr_algorithmtype   algorithm );
```

Description Defines the value list for a master attribute and selects the algorithm used to pick values from this list.

CLI Equivalent

```
BestPprMAttrVariationSet attribute=<attribute>
value_list=<value_list> algorithm=<algorithm>
```

CLI Abbreviation

```
pprmavs attr=<attribute> list=<value_list> alg=<algorithm>
```

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

attribute Master attribute; see *"b_mattrproptype" on page 251*.

value_list The values in this list are permutated according to the selected algorithm. For the syntax of the list, see *"Numeric values" on page 299*.

algorithm Determines the algorithm used to pick values from the value list. See *"bppr_algorithmtype" on page 292*.

See also *"BestPprMAttrVariationDefaultSet" on page 205*
"BestPprMAttrVariationGet" on page 206

PPR Report Functions

The following functions are used to set the report properties and to generate reports:

Function	Result
"BestPprReportPropDefaultSet" on page 209	Sets all report properties to default values.
"BestPprReportPropSet" on page 211	Sets a report property.
"BestPprReportPropGet" on page 210	Reads the value of a report property.
"BestPprReportWrite" on page 212	Generates a report.
"BestPprReportFile" on page 209	Writes a report to a file.
"BestPprReportDelete" on page 208	Frees memory used for a generated report.

How to use the functions is described in *"Generating PPR Reports"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestPprReportDelete

Call

```
b_errtype BestPprReportDelete( b_handletype handle );
```

Description Frees the memory space allocated by *"BestPprReportWrite" on page 212*.

CLI Equivalent `BestPprReportDelete`

CLI Abbreviation `pprrd`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also –

BestPprReportFile

Call

```
b_errtype BestPprReportFile(
      b_handletype    handle,
      b_charptrtype   filename );
```

Description Generates a report and writes the content to the specified file. If the file already exists, it will be overwritten. For more information on reports, see *"BestPprReportWrite" on page 212*.

CLI Equivalent `BestPprReportFile filename=<filename>`

CLI Abbreviation `pprrf file=<filename>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

filename String holding a valid filename for the report file.

See also –

BestPprReportPropDefaultSet

Call `b_errtype BestPprReportPropDefaultSet( b_handletype handle );`

Description Sets the report properties to default values. For a description of properties and default values, see *"bppr_reportproptype" on page 303*.

CLI Equivalent `BestPprReportPropDefaultSet`

CLI Abbreviation `pprrpds`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

See also
"BestPprReportPropGet" on page 210
"BestPprReportPropSet" on page 211

BestPprReportPropGet

Call

```
b_errtype BestPprReportPropGet(
      b_handletype          handle,
      bppr_reportproptype   prop,
      b_int32               *value );
```

Description Reads the value of a report property. The report properties determine which information is printed to the report.

CLI Equivalent `BestPprReportPropGet prop=<prop>`

CLI Abbreviation `pprrpg p=<prop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

prop Property to be read; see *"bppr_reportproptype" on page 303*.

Output Parameters

value Value of the read report property; see *"bppr_reportproptype" on page 303*.

See also

"BestPprReportPropDefaultSet" on page 209
"BestPprReportPropSet" on page 211

BestPprReportPropSet

Call

```
b_errtype BestPprReportPropSet(
      b_handletype          handle,
      bppr_reportproptype   prop,
      b_int32               value );
```

Description Sets a report property. The report properties determine which information is printed to the report.

CLI Equivalent `BestPprReportPropSet prop=<prop> value=<value>`

CLI Abbreviation `pprrps p=<prop> val=<value>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

prop Property to be set; see *"bppr_reportproptype" on page 303*.

value Value to which the property is set; see *"bppr_reportproptype" on page 303*.

See also *"BestPprReportPropDefaultSet" on page 209*
"BestPprReportPropGet" on page 210

BestPprReportWrite

Call

```
b_errtype BestPprReportWrite(
      b_handletype   handle,
      b_charptrtype  *report_ptr );
```

Description

Generates a report. This function performs the permutations using the properties of master block variations, master attributes and target attributes, and reports the results.

Block or attribute pages do not need to be generated and downloaded to the testcard before calling this function. The permutations are performed internally, nothing is downloaded to or uploaded from the testcard.

The contents of the report can be determined with *"BestPprReportPropSet" on page 211*. The report may also contain hints and warnings, which are described in the table below.

NOTE The function returns a pointer to the generated report string. The required memory space is allocated automatically. When the report is no longer needed, the memory space must be freed with *"BestPprReportDelete" on page 208*.

CLI Equivalent

`BestPprReportWrite.`

The report string will be displayed in the CLI window.

CLI Abbreviation `pprrw`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

Output Parameters **report_ptr** Pointer to the report string.

NOTE If this function is used twice with the same handle, the memory of a previously generated report is automatically deleted.

See also –

Hints and Warnings in the Report String

Text in the Report String	Description
HINT: Coverage assumes that all permutation numbers from 1 to first permutation = <N> have been executed before.	Issued if the first permutation number is not equal to 1.
HINT: This master attribute page should be called with a block size = <N> dwords to avoid shortened bursts.	Always issued. Informs you about the minimum block size to be used if the master attribute page is downloaded and called.
WARNING: Burstlengths cannot be guaranteed, because the largest block size (<N> dwords) is smaller than the sum of all burstlengths (<M>).	Issued if the parameter BPPR_BLK_SIZE is smaller than the sum of all burstlengths N and therefore the performance of all burstlengths cannot be guaranteed.
WARNING: There is no permutation of block parameters, which allows the usage of the MWI command according to PCI Specification.	The variation properties for block variation parameters are defined in a way that no MWI command can be generated by the PCI PPR software. See *"BestPprBlockVariationSet" on page 199*.
WARNING: MWI bursts must have sizes of multiple cachelines. Therefore make sure you use infinite burstlength when generating attribute pages by PPR, or set up your own attribute page so that a LAST bit will not interrupt transfer within a cacheline!	Burstlengths are set to values shorter than cacheline size, and simultaneously MWI is contained in the value list of commands. This will result in MWI transfers being interrupted within a cacheline.

Target Attribute Permutation Functions

The following functions are used to prepare and perform the permutation of the target attributes:

Function	Result
"BestPprTAttrInit" on page 216	Initializes the target attribute permutator.
"BestPprTAttrPermPropDefaultSet" on page 217	Sets all target attribute properties to default values.
"BestPprTAttrPermPropSet" on page 218	Sets values of target attribute permutation properties.
"BestPprTAttrPermPropGet" on page 217	Reads values of target attribute permutation properties.
"BestPprTAttrVariationDefaultSet" on page 219	Sets all target attribute variation parameters to default values.
"BestPprTAttrVariationSet" on page 221	Sets the value list and the corresponding algorithm of a target attribute.
"BestPprTAttrVariationGet" on page 220	Reads the value list and the corresponding algorithm of a target attribute.
"BestPprTAttrGenerate" on page 216	Generates a target attribute page.
"BestPprTAttrResultGet" on page 219	Reads result parameters.
"BestPprTAttrCoverageGet" on page 215	Returns coverage and repetition length.

How to use the functions is described in *"Programming Target Attribute Permutations"* in the *Agilent E2940A Opt. 320 C-API/PPR Programmer's Guide*.

BestPprTAttrCoverageGet

Call

```
b_errtype BestPprTAttrCoverageGet(
      b_handletype      handle,
      b_int32           nmb_attr,
      b_tattrproptype   *attr_list,
      b_int32           *rep_len,
      b_int32           *covered );
```

Description Internally performs the permutation algorithm with the parameters currently set. “Internally” means that nothing is downloaded to the testcard. This function determines whether all permutations of a given tuple of target attributes are covered, and returns the repetition length of the tuple.

CLI Equivalent No CLI equivalent. To get coverage information with the CLI, use *“BestPprReportWrite” on page 212*.

CLI Abbreviation No CLI abbreviation.

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

nmb_attr Number of target attributes in the tuple.

attr_list This array defines the tuple with the coverage to be tested. The tuple holds nmb_attr entries. The entries must be valid values of *“b_tattrproptype” on page 278*.

Output Parameters

rep_len Returns the repetition length of the tuple (that is the number of permutations of the tuple).

covered Returned values:

- 1 – if the tuple is covered after complete execution of the target attribute page created by the permutation algorithm.
- 0 – if this is not the case.

See also –

BestPprTAttrGenerate

Call `b_errtype BestPprTAttrGenerate( b_handletype handle);`

Description Generates a target attribute page using the properties shown in the table below. The target attribute page is downloaded to the testcard.

Properties of Target Attribute Page Generation	Adjusted by Function
Generic setup properties	*"BestPprGenPropSet" on page 190*
Target attribute permutation properties	*"BestPprTAttrPermPropSet" on page 218*
Target attribute variation parameters	*"BestPprTAttrVariationSet" on page 221*

CLI Equivalent `BestPprTAttrGenerate`

CLI Abbreviation `pprtag`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"b_tattrproptype" on page 278*

BestPprTAttrInit

Call `b_errtype BestPprTAttrInit( b_handletype handle );`

Description Initializes the target attribute permutator and sets all of its properties to default values.

To set individual properties to default values, use either the function *"BestPprTAttrPermPropDefaultSet" on page 217* or *"BestPprTAttrVariationDefaultSet" on page 219*.

CLI Equivalent `BestPprTAttrInit`

CLI Abbreviation `pprtai`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also –

BestPprTAttrPermPropDefaultSet

Call

```
b_errtype BestPprTAttrPermPropDefaultSet( b_handletype handle );
```

Description Sets the target attribute permutation properties to default values. For a description of properties and default values, see *"bppr_tattrpermproptype" on page 304*.

CLI Equivalent `BestPprTAttrPermPropDefaultSet`

CLI Abbreviation `pprbvds`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestPprTAttrPermPropGet" on page 217*
"BestPprTAttrPermPropSet" on page 218

BestPprTAttrPermPropGet

Call

```
b_errtype BestPprTAttrPermPropGet(
      b_handletype            handle,
      bppr_tattrpermproptype  prop,
      b_int32                 *value );
```

Description Reads the value of a target attribute permutation property.

CLI Equivalent `BestPprTAttrPermPropGet prop=<prop>`

CLI Abbreviation `pprtappg p=<prop>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

prop Property to be read; see *"bppr_tattrpermproptype" on page 304*.

Output Parameters **value** Value of the read property; see *"bppr_tattrpermproptype" on page 304*.

See also *"BestPprTAttrPermPropDefaultSet" on page 217*
"BestPprTAttrPermPropSet" on page 218

BestPprTAttrPermPropSet

Call

```
b_errtype BestPprTAttrPermPropSet(
      b_handletype             handle,
      bppr_tattrpermproptype   prop,
      b_int32                  value );
```

Description Sets a target attribute permutation property.

CLI Equivalent `BestPprTAttrPermPropSet prop=<prop> value=<value>`

CLI Abbreviation `pprtapps p=<prop> val=<value>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

prop Property to be set; see *“bppr_tattrpermproptype” on page 304*.

value Value to which the property is set; *“bppr_tattrpermproptype” on page 304*.

See also *“BestPprTAttrPermPropDefaultSet” on page 217*
“BestPprTAttrPermPropGet” on page 217

BestPprTAttrResultGet

Call

```
b_errtype BestPprTAttrResultGet(
      b_handletype                handle,
      bppr_tattrresultparamtype   param,
      b_int32                     *value );
```

Description Internally performs the permutation algorithm with the parameters currently set. “Internally” means nothing is downloaded to the testcard. This function reads the value of a selected result parameter.

CLI Equivalent `BestPprTAttrResultGet param=<param>`

CLI Abbreviation `pprtarg p=<param>`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

param Result parameter to be read; see *“bppr_tattrresultparamtype” on page 305*.

Output Parameters

value Value of the read result parameter; see *“bppr_tattrresultparamtype” on page 305*.

See also –

BestPprTAttrVariationDefaultSet

Call

```
b_errtype BestPprTAttrVariationDefaultSet( b_handletype handle );
```

Description Sets the target attribute variation parameters to default values. For a description of parameters and default values, see *“b_tattrproptype” on page 278*.

CLI Equivalent `BestPprTAttrVariationDefaultSet`

CLI Abbreviation `pprtavds`

Return Value Error code; see *“b_errtype” on page 241*.

Input Parameters

handle Handle to identify the session.

See also *“BestPprTAttrVariationGet” on page 220*
“BestPprTAttrVariationSet” on page 221

BestPprTAttrVariationGet

Call

```
b_errtype BestPprTAttrVariationGet(
      b_handletype         handle,
      b_tattrproptype      attribute,
      b_charptrtype        *value_list,
      bppr_algorithmtype   *algorithm );
```

Description Reads variation properties of a target attribute.

CLI Equivalent `BestPprTAttrVariationGet attribute=<attribute>`

CLI Abbreviation `pprtvg attr=<attribute>`

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

attribute Target attribute to be read; see *"b_tattrproptype" on page 278*.

Output Parameters

value_list Returns a pointer to a string containing a comma-separated list of values. The values in this list are permutated according to the selected algorithm. For a description of the string's contents, refer to *"Numeric values" on page 299*.

If the PCI PPR software has not yet been initialized with *"BestPprBlockInit" on page 194* or *"BestPprInit" on page 191*, the list will be empty and an error will be returned.

algorithm Returns the algorithm used for this attribute. See *"bppr_algorithmtype" on page 292*.

See also

"BestPprTAttrVariationDefaultSet" on page 219
"BestPprTAttrVariationSet" on page 221

BestPprTAttrVariationSet

Call

```
b_errtype BestPprTAttrVariationSet(
      b_handletype         handle,
      b_tattrproptype      attribute,
      b_charptrtype        value_list,
      bppr_algorithmtype   algorithm );
```

Description Defines the value list for a target attribute and selects the algorithm used to pick values from this list.

CLI Equivalent

```
BestPprTAttrVariationSet attribute=<attribute>
value_list=<value_list> algorithm=<algorithm>
```

CLI Abbreviation

```
pprtavs attr=<attribute> list=<value_list> alg=<algorithm>
```

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters

handle Handle to identify the session.

attribute Target attribute; see *"b_tattrproptype" on page 278*.

value_list List of values to be permutated according to the selected algorithm. For the syntax of the list, see *"Numeric values" on page 299*.

algorithm Algorithm used to pick values from the value list; see *"bppr_algorithmtype" on page 292*.

See also

"BestPprTAttrVariationDefaultSet" on page 219
"BestPprTAttrVariationGet" on page 220

Error Handling

The application programming interface of the testcard provides several functions to handle occured errors.

Error functions can be used to query the error code and the error string of the last error that occured in the specified session. But it is also possible to query the meaning of an error code if no handle is available.

Error Functions

The following functions are used for error handling:

Function	Result
"BestLastErrorGet" on page 224	Returns the last error.
"BestLastErrorStringGet" on page 224	Returns the error string of the last error.
"BestErrorStringGet" on page 225	Returns the error string of an error.

BestLastErrorGet

Call `b_errtype BestLastErrorGet( b_handletype handle );`

Description Returns the code of the last error that occurred with the handle.

CLI equivalent No CLI equivalent.

CLI abbreviation No CLI abbreviation.

Return Value Error code; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestLastErrorStringGet" on page 224*
"BestErrorStringGet" on page 225

BestLastErrorStringGet

Call `b_charptrtype BestLastErrorStringGet( b_handletype handle );`

Description Returns the error string of the last error that occurred with the handle.

CLI equivalent No CLI equivalent.

CLI abbreviation No CLI abbreviation.

Return Value Error string; see *"b_errtype" on page 241*.

Input Parameters **handle** Handle to identify the session.

See also *"BestLastErrorGet" on page 224*
"BestErrorStringGet" on page 225

BestErrorStringGet

Call `b_charptrtype BestErrorStringGet( b_errtype err );`

Description Returns the error string of an error. Use this function if no handle is available (for example, if *“BestOpen” on page 21* fails).

CLI equivalent No CLI equivalent.

CLI abbreviation No CLI abbreviation.

Return Value Error string; see *“b_errtype” on page 241*.

Input Parameters **err** Error define.

See also *“BestLastErrorGet” on page 224*
“BestLastErrorStringGet” on page 224

Type Definitions

All type definitions are listed in alphabetical order.

Several definitions differ in there naming conventions. So in the first part the type names begin with “b_”, in the second part the names begin with “bppr_”. The suffix “ppr” identifies type definitions which are only used in Protocol Permutator and Randomizer functions.

For further information on the naming conventions, see *“Conventions” on page 14*.

b_addrspacetype

addrspace (CLI Abbreviation)	Description
B_ASP_CONFIG (config)	Type 0 access to config space.
B_ASP_CONFIG_TYPE1 (configtype1)	Type 1 access to config space.
B_ASP_IO (io)	Access to I/O space.
B_ASP_MEM (mem)	Access to memory space.

b_blkproptype

blk_prop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_BLK_ATTRPAGE (apage)	0 … 63; **default:** 0 – if page size is 4 , or 0 … 7; **default:** 0 – if page size is 32 (for compatibility reasons).	Pointer to a master attribute page to define PCI protocol behavior for the block transfer.
B_BLK_BUSADDR (bad)	32 bits.	PCI bus address, used as the starting address for the block transfer (64-bit address width only if master block property B_BLK_BUSDAC is enabled).
B_BLK_BUSADDR_HI (badhi)	Upper 32 bits of a 64-bit address.	
B_BLK_BUSCMD (cmd)	PCI bus command for the block transfer (C/BE#[3:0] during address phases).	
	B_CMD_INT_ACK (int_ack)	Interrupt Acknowledge
	B_CMD_SPECIAL (special)	Special Cycle
	B_CMD_IO_READ (io_read)	I/O Read
	B_CMD_IO_WRITE (io_write)	I/O Write
	B_CMD_RESERVED_4 (reserved_4)	Reserved
	B_CMD_RESERVED_5 (reserved_5)	Reserved
	default: B_CMD_MEM_READ (mem_read)	Memory Read
	B_CMD_MEM_WRITE (mem_write)	Memory Write
	B_CMD_RESERVED_8 (reserved_8)	Reserved
	B_CMD_RESERVED_9 (reserved_9)	Reserved
	B_CMD_CONFIG_READ (config_read)	Configuration Read

blk_prop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_BLK_BUSCMD (cmd)	B_CMD_CONFIG_WRITE (config_write)	Configuration Write
	B_CMD_MEM_READMULTIPLE (mem_readmultipl)	Memory Read Multiple (MRM)
	B_CMD_MEM_READLINE (mem_readline)	Memory Read Line (MRL)
	B_CMD_MEM_WRITEINVALIDATE, (writeinvalidate)	Memory Write & Invalidate (MWI) MWI must be enabled in the configuration space and must start at a cacheline boundary. If not, it is replaced by Memory Write. **Note:** When used in combination with B_M_LAST, illegal burstlengths could be generated.
B_BLK_BUSDAC (dac)	Determines whether 64-bit or 32-bit addresses are transferred during an address phase.	
	default: 0	Single address cycle with 32-bit address.
	1	Dual address cycle with 64-bit address. Uses B_BLK_BUSADDR_HI.
B_BLK_BYTEN (ben)	0x00 … 0xFF **default:** 00	Pointer to the byte enable memory. 00 – all byte enables are active during all data phases of the block transfer. 1 … 15 – values for the lower byte enables (C/BE[3:0]). Valid for all data phases of the block transfer. (The upper byte enables are set to the same value.) 16 … 255 – pointer to the byte enable memory storing a series of byte enables (8 bit) that can be worked through during the block transfer.
B_BLK_BYTEN_VAR (benv)	Determines whether the byte enables are fixed or variable during the block transfer.	
	default: 0	Byte enables are fixed. The pointer to the byte enable memory is not changed.
	1	Byte enables are variable. The next byte enables are taken from byte enable memory after each successful data transfer.
B_BLK_COMPFLAG (cflag)	The compare flag determines whether read data are compared with data stored in the testcard's internal data memory. The comparison takes place while the data is read. The read data is not stored in the memory.	
	default: 0	No comparison. Data is stored. Start address of internal memory is given by B_BLK_INTADDR.
	1	Data is compared with data stored in internal memory, beginning with the start address given by B_BLK_COMPOFFS.
B_BLK_COMPOFFS (coffs)	0 … 0x1FFFC **default:** 0	Dword-aligned compare offset, see B_BLK_COMPFLAG.

blk_prop (CLI Abbreviation)	**value (CLI Abbreviation)**	**Description**
B_BLK_CONDSTART (cond)	The conditional start flag determines whether the master run of a block is performed immediately or whether a start condition must first be fulfilled.	
	default: 0	Start immediately.
	1	Wait until start condition is fulfilled. The start condition must be set using *"BestMasterCondStartPattSet" on page 108*. The generic master property B_MGEN_RUNMODE must be set at 1. If not, conditional start is generally disabled.
B_BLK_CONTATTR (contattr)	The continue attributes flag determines whether an attribute page is restarted after a block transfer has been completed or whether transfer is continued from the current pointer position. After startup or reset, the master attribute page is always started at the beginning. **Note:** If this flag is 0, it can still be overridden by the setting of the generic master property B_MGEN_ATTRMODE. See *"BestMasterGenPropSet" on page 107*.	
	default: 0	The master attribute page as given by B_BLK_ATTRPAGE is restarted.
	1	Transfer is continued from the current pointer position.
B_BLK_INTADDR (iad)	00000\h … FFFFF\h **default:** 0 Must be dword-aligned and have the same alignment as B_BLK_BUSADDR.	Internal Address of the testcard's data memory. Used to access data to be written or to store data to be read. (NOT used for data compare!)

b_boardproptype

boardprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_BOARD_HOTSWAPMODE (hsmode)	Hot Swap Mode. For information on hot swapping, refer to the *CompactPCI Hot Swap Specification*. To set the card's hot swap mode, use *"BestBoardPropSet" on page 35*. To make the card start with this hot swap mode at the next power up, you have to additionally use *"BestBoardPropStore" on page 36*.	
	B_BOARD_HOTSWAP_NONE (hsnone)	Non hot swap
	B_BOARD_HOTSWAP_BASIC (hsbasic)	Basic hot swap
	B_BOARD_HOTSWAP_FULL (hsfull)	Full hot swap
	B_BOARD_HOTSWAP_HIGH (hshigh)	High availability
	B_BOARD_HOTSWAP_STEALTH (hsstealth)	Stealth mode to conceal the card from the system. Same as setting the jumper as described in the *Agilent E2940A PCI Exerciser and Analyzer User´s Guide* (pdf file).
B_BOARD_PERREN (perren)	Enables/disables parity error (PERR#) reporting by setting the configuration space command register bit 6.	
	default: 0	Disables parity error reporting.
	1	Enables parity error reporting.
B_BOARD_RSTMODE (rstmode)	B_RSTMODE_RESETALL (resetall, 0)	After a PCI reset, a testcard reset is performed.
	B_RSTMODE_RESETSM (resetsm, 1)	After a PCI reset, the master, target, and analyzer state machines are reset, without affecting configuration space, decoders or other onboard properties.
	default: B_RSTMODE_RESETCONFIG (resetconfig, 2)	After a PCI reset, the configuration space is reset. The read/write bits are restored according to the setting of B_PU_CONFRESTORE (see *"BestPowerUpPropSet" on page 38*).

boardprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_BOARD_SERREN (serren)	Enables/disables parity system error (SERR#) reporting by setting the configuration space command register bit 8.	
	default: 0	Disables system error reporting.
	1	Enables system error reporting.

b_cpcipintype

pin (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_CPCI_BDSEL (bdsel)	0, 1 **default:** 0	State of the BD_SEL# signal. **Note:** Setting the signal to "1" switches the card off.
B_CPCI_ENUM (enum)	0, 1 **default:** 0	State of the ENUM# signal.
B_CPCI_HEALTHY (healthy)	0, 1 **default:** 0	State of the HEALTHY# signal.

b_cpcistatustype

status (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_CPCI_EIM (eim)	ENUM# signal mask.	
	default: 0	Enable signal
	1	Disable (mask) signal
B_CPCI_EXT (ext)	0, 1 **default:** 0	1 = the testcard is about to be extracted
B_CPCI_INS (ins)	0, 1 **default:** 0	1 = the testcard has just been inserted
B_CPCI_LOO (loo)	Hot swap LED Control bit (blue LED).	
	default: 0	LED off
	1	LED on

b_decodertype

decoder (CLI Abbreviation)	Description
B_DEC_CONFIG (config)	Configuration decoder
B_DEC_EXPROM (exprom)	Expansion ROM decoder
B_DEC_FULL_CONFIG (fullconfig)	Full configuration decoder (configuration cycle type 1 decoder)
B_DEC_STANDARD_1 (std1)	Standard decoder 1
B_DEC_STANDARD_2 (std2)	Standard decoder 2
B_DEC_STANDARD_3 (std3)	Standard decoder 3
B_DEC_STANDARD_4 (std4)	Standard decoder 4
B_DEC_STANDARD_5 (std5)	Standard decoder 5
B_DEC_STANDARD_6 (std6)	Standard decoder 6
B_DEC_SUBTRACTIV (subtract)	Subtractive decoder

b_decproptype

The target decoder properties of type b_decproptype can be grouped according to their functions:

- *"Decoding Properties" on page 235*
- *"Info Properties" on page 239*
- *"Resource Properties" on page 240*

Decoding Properties

<table>
<tr><th>decoder_prop
(CLI Abbreviation)</th><th>Affects Decoder</th><th>value
(CLI Abbreviation)</th><th>Description</th></tr>
<tr><td>B_DEC_BASEADDR
(base)</td><td>All except Subtractive</td><td>Bits[31::4] of a 32-bit value</td><td>Base address of the decoder and also the entry of the Base Address Register of the decoder in the configuration space.

Specific bits have specific meanings during configuration cycles. If you change their values while the testcard is being used as a target, the behavior of the device/system under test can change.

Warning: Setting multiple devices with overlapping address ranges within a system may result in hardware damage.</td></tr>
<tr><td>B_DEC_BASEADDR_HI
(basehi)</td><td>All except Subtractive</td><td>32 Bit</td><td>Upper 32-bit of a 64-bit base address.
See B_DEC_BASEADDR.

B_DEC_DAC must be set to B_DAC_YES.</td></tr>
<tr><td>B_DEC_BASEDEC
(basedec)</td><td>Standard</td><td>1 … 6</td><td>Base decoder identifier. To be used with overlay behavior.

Note: The decoders 1 … 6 provide increasing priority.
Therefore, the decoder with overlay behavior must have a higher number than the base decoder. Otherwise the base decoder cannot decode.</td></tr>
<tr><td rowspan="5">B_DEC_BEHAVIOR
(behavior)</td><td colspan="3">Determines the behavior of Standard Decoders 1 … 6.
This property influences the availability and ranges of other target decoder properties.</td></tr>
<tr><td>Standard</td><td>B_BEH_NORMAL
(normal)</td><td>Decoder behaves PC compliant, according to the settings in the configuration space.</td></tr>
<tr><td></td><td>B_BEH_OVERLAY
(overlay)</td><td>Decoder is an overlay decoder. All bits of its base address register return 0 on a read.

B_DEC_BASEDEC must contain the base decoder identifier.</td></tr>
<tr><td></td><td>B_BEH_CONFIG
(config)</td><td>Decoder behaves like a configuration decoder. All bits of its base address register return 0 on a read.</td></tr>
<tr><td></td><td>B_BEH_CUSTOM
(custom)</td><td>Decoder can be set up non-PCI compliant. Property values are not checked (see “BestTargetDecoderProg” on page 139).</td></tr>
</table>

decoder_prop (CLI Abbreviation)	Affects Decoder	value (CLI Abbreviation)	Description
B_DEC_BUSCMD (cmd)	Commands to be decoded. Use a logical OR-connection to decode multiple commands .		
	All except Expan. ROM	B_CMDBIT_INT_ACK (0x0001)	Interrupt Acknowledge
		B_CMDBIT_SPECIAL (0x0002)	Special Cycle
		B_CMDBIT_IO_READ (0x0004)	I/O Read
		B_CMDBIT_IO_WRITE (0x0008)	I/O Write
		B_CMDBIT_RESERVED_4 (0x0010)	Reserved
		B_CMDBIT_RESERVED_5 (0x0020)	Reserved
		B_CMDBIT_MEM_READ (0x0040)	Memory Read
		B_CMDBIT_MEM_WRITE (0x0080)	Memory Write
		B_CMDBIT_RESERVED_8 (0x0100)	Reserved
		B_CMDBIT_RESERVED_9 (0x0200)	Reserved
		B_CMDBIT_CONFIG_READ (0x0400)	Configuration Read
		B_CMDBIT_CONFIG_WRITE (0x0800)	Configuration Write
		B_CMDBIT_MEM_ READMULTIPLE (0x1000)	Memory Read Multiple (MRM)
		B_CMDBIT_MEM_READLINE (0x4000)	Memory Read Line (MRL)
		B_CMDBIT_MEM_ WRITEINVALIDATE (0x8000)	Memory Write & Invalidate (MWI) MWI must be enabled in the configuration space and must start at a cacheline boundary. Otherwise “Memory Write” replaces MWI.

decoder_prop (CLI Abbreviation)	Affects Decoder	value (CLI Abbreviation)	Description
B_DEC_DAC (dac)	Specifies whether the decoder is 64-bit capable. For 64-bit address decoding, two standard decoders are used as a pair: Standard decoder 1 decodes the lower 32 bits, standard decoder 2 the upper 32. Standard decoder 3 decodes the lower 32 bits, standard decoder 4 the upper 32. Standard decoder 5 decodes the lower 32 bits, standard decoder 6 the upper 32.		
	Standard Subtractive	B_DAC_NO (no, 0)	Decoder is not 64-bit capable.
		B_DAC_YES (yes, 1)	Decoder is 64-bit capable.
	Subtractive	B_DAC_BOTH (both, 2)	Decodes both 32 and 64-bit addresses.
B_DEC_IDSEL (idsel)	Specifies whether IDSEL must be asserted to recognize a configuration command.		
	Standard Configuration Subtractive	B_IDSEL_ASSERT (assert)	IDSEL must be asserted.
		B_IDSEL_DEASSERT (deassert)	IDSEL must be de-asserted.
		B_IDSEL_DONTCARE (dontcare)	IDSEL is ignored.
B_DEC_MASK (mask)	Standard	32 Bit	Specifies the bits in the base address register relevant for decoding: 1 – read/write (relevant) 0 – read-only (not relevant) **Warning:** Setting multiple devices with overlapping address ranges within a system may result in hardware damage.
B_DEC_MASK_HI (maskhi)	Standard	32 Bit	Upper 32-bit of a 64-bit address. See B_DEC_MASK. B_DEC_DAC must be set to B_DAC_YES or B_DAC_BOTH.
B_DEC_PRIMARY_BUS (primary)	Config. Type 1	0 … 255	Specifies the lowest bus number for which type 1 configuration cycles are decoded.
B_DEC_SECONDARY_BUS (secondary)	Config. Type 1	0 … 255	Specifies the highest bus number for which type 1 configuration cycles are decoded.
B_DEC_SIZE (size)	Standard	0, 5 … 64	Determines the size of the decoded address range (actual size = $2^{B\_DEC\_SIZE}$). A size of 0 switches off the decoder by setting all bits of its base address register to zero, unless behavior is set to “custom”. In this case, you have to set them to zero with B_DEC_BASEADDR. **Warning:** Setting multiple devices with overlapping address ranges within a system may result in hardware damage.

decoder_prop (CLI Abbreviation)	Affects Decoder	value (CLI Abbreviation)	Description
B_DEC_SPEED (speed)	Determines the decode speed (that is the delay until DEVSEL# is asserted after request). Fast decode has the following restrictions: **Resource** must be the data memory or the compare unit. For **memory decoders**, the size value must be set between 20 and 43 (or mask must be set accordingly). “Memory Read Multiple” is not allowed. For **I/O decoders**, the size value must be between 2 and 16 (or mask must be set accordingly). **Configuration decoders** require IDSEL to be set. Only the bits carrying the “function” information are decoded. **Expansion ROM decoders and Full configuration decoders** cannot fast decode. **Bursts** may run out of range. At one time, only **one decoder of each type** (standard, config, subtractive) can do fast decoding.		
	Standard Configuration Subtractive	B_DSP_FAST (fast)	Sets the decode speed to fast.
	Standard Configuration Subtractive	B_DSP_PROTECTEDFAST (pfast)	Sets the decode speed to fast and performs a range check. This forces single cycles.
	All	B_DSP_MEDIUM (medium)	Sets double speed to medium.
		B_DSP_SLOW (slow)	Sets decode speed to slow.
		B_DSP_NODEVSEL (nodevsel)	The decoder never asserts DEVSEL#. **Note:** The transaction can still be claimed by a subtractive decoder.

Info Properties

decoder_prop (CLI Abbreviation)	Affects Decoder	value (CLI Abbreviation)	Description
B_DEC_LOCATION (loc)	Sets the bits in the configuration space that specify the location where the BIOS has to place the memory range.		
	Standard	B_LOC_SPACE32 (space32)	In 32-bit memory address range.
		B_LOC_BELOW1MEG (below1meg)	Below 1 MB memory address range.
		B_LOC_SPACE64 (space64)	Location is a 64-bit memory address range.
		B_LOC_IO (io)	Location is an I/O address range.
B_DEC_PREFETCH (prefetch)	Sets the “prefetch” bit in the configuration space. This specifies that a device provides “prefetchable” memory.		
	Standard	**default:** 0	“Prefetch” bit is not set.
		1	“Prefetch” bit is set.

Resource Properties

decoder_prop (CLI Abbreviation)	Affects Decoder	value (CLI Abbreviation)	Description
B_DEC_RESBASE (resbase)	All	32 Bit	Internal Base Address of the resource. The value must be a multiple of the resource size ($2^{B\_DEC\_RESSIZE}$).
B_DEC_RESOURCE (res)	Specifies the internal resource to which the decoder is connected.		
	All	B_RES_DATA (data)	Internal data memory. Protocol attributes are used as defined in the target attribute page.
		B_RES_DATA_ DEFATTR (datadef)	Internal data memory. Default protocol attributes are used.
		B_RES_COMPARE (comp)	Compare unit. For write transactions only. Incoming data is not stored. Protocol attributes are used as defined in the target attribute page.
		B_RES_COMPARE_ DEFATTR (compdef)	Compare unit. For write transactions only. Incoming data is not stored. Default protocol attributes are used.
		B_RES_REGFILE (regfile)	Configuration Space, public and private section (this is also referred to as “internal register file”). **Note:** The private section is reserved. Overwriting can result in the loss of the connection to the testcard.
		B_RES_STATICIO (staticio)	Static I/O.
B_DEC_RESSIZE (ressize)	All	0 … 64	Determines the size of the resource (actual size of the address range = $2^{B\_DEC\_RESSIZE}$).

b_errtype

There are two types of error codes:

- **Software Errors** (codes beginning with “B_E_”).

  The software errors are issued by the testcard software running on the control PC.

- **Firmware Errors** (codes beginning with “B_EFW_”).

  Firmware errors are issued by the firmware running directly on the card.

This information may help you to localize some errors.

Software Errors

err/Error Code	Error String	Notes/Recommendations/Help
B_E_ALIGN	Parameter *<parameter>* must be aligned to *<alignmt>*. Its value is *<value>*.	
B_E_BAD_DECODER_NUMBER	No valid decoder number.	
B_E_BAD_FILE_FORMAT	Bad format for file *<filename>*.	
B_E_BAD_HANDLE	Bad handle. You may only use handles obtained from ”BestOpen()”.	
B_E_BAUDRATE	Could not set new baud rate.	
B_E_BOARD_RESET	Could not re-connect after board reset.	
B_E_CANNOT_CONNECT	The specified port *<port>* is unable to connect. Please check cable and connectors and try again. If your card is E2925A it may be connected exclusively.	Another reason may be that the card is busy, for example, with transferring data via the fast host interface.
B_E_CANNOT_CONNECT_CORE	*<port>* port is unable to connect because the card is operating in core mode. Please reset the card. If the problem persists, run a hardware update.	
B_E_CANNOT_CONNECT_EXCLUSIVE	The specified port *<port>* cannot get an exclusive connection to the card. Check that no other port is connected exclusively and try again. If the problem persists, try a conventional (non-exclusive) connection. This message applies to the E2925A only.	Another reason may be that the card is busy, for example, with transferring data via the fast host interface.
B_E_CONF_REG	Could not write to config space OR the value to be written is invalid for that register.	
B_E_CONFIG_MASK_INVALID	Mask *<mask>* for config space register *<cfreg>* is invalid or not PCI compliant.	

err/Error Code	Error String	Notes/Recommendations/Help
B_E_CONFIG_VALUE_INVALID	Value *<value>* for config space register *<cfreg>* is invalid or not PCI compliant.	
B_E_CONNECTION_LOST	*<port>* port lost its connection. Please check cable connection, and try to close and re-open the connection.	
B_E_CONNECTION_LOST_CMD	*<port>* port lost its connection. Please check cable connection and try to close and re-open the connection. *<first cmdbyte><second cmdbyte>*	
B_E_CORE_VERSION_MISMATCH	Minimal core version for firmware *<version1>* is *<version2>*. Current core version is *<version3>*.	Try hardware update from graphical user interface.
B_E_CPU_MISALIGNED	CPU Port address misaligned.	
B_E_DEC_BASE_NOT_0	Base address must be 0 when size is 0.	
B_E_DEC_BASE_WRITE	Error while writing base address into hardware.	Base address is not PCI compliant.
B_E_DEC_CHECK	Unknown decoder error. (Perhaps there is no PCI clock or hardware is busy.)	
B_E_DEC_CONFIG_ACCESS	Error while accessing hardware (config space busy, or no PCI clock).	
B_E_DEC_INVALID_MODE	Invalid mode specified for this decoder.	
B_E_DEC_INVALID_SIZE	Invalid size specified for this decoder.	
B_E_DEC_ROM_SIZE_0_ENABLED	Inconsistency: Generic ROM enable active, but size is 0.	
B_E_DEC_SIZE_BASE_MISMATCH	There is a decoder size and base address mismatch.	
B_E_DRIVER_VERSION_DIFF	The PCI driver version is incompatible with this C-API.	Re-install the software.
B_E_DYNAMIC_CAPABILITY	Error while checking dynamic capabilities.	
B_E_ERROR	Error during command transfer.	
B_E_FCT_PARAM	Function *<function>* rejected parameter *<param>* : *<value>*.	
B_E_FILE_OPEN	Could not open file *<filename>*.	
B_E_FUNC	Functional onboard error.	Runtime error on the hardware. Try again, if the problem persists, reboot the card.
B_E_HIF_OPEN	Could not open BEST Fast Host Interface driver.	
B_E_HOST_MEM_FULL	Not enough memory.	
B_E_HW_BUSY	Board hardware is busy, or no PCI clock.	
B_E_INVALID_OBS_RULE	Observer rule unknown or not implemented.	
B_E_MASK_REG	Could not write config mask OR the mask value is invalid for that register.	
B_E_MASTER_ABORT	Master abort occurred, no target response.	
B_E_NO_BEST_PCI_DEVICE_FOUND	Specified device not found on PCI Bus.	

err/Error Code	Error String	Notes/Recommendations/Help
B_E_NO_HANDLE_LEFT	Cannot open another port because no handle is left. Close some ports and try again.	
B_E_NO_PCI_CLOCK	No or slow PCI clock.	
B_E_NOT_COMPACT	This command is not supported by the E2940A.	
B_E_NOT_CONNECTED	The specified port <*port*> is not connected. Please execute the BestConnect() command. This error message applies to the E2925A only.	
B_E_NOT_E2925A	This command is not supported by the E2925A.	
B_E_OK	No error.	
B_E_ONLY_COMPACT	This command is only supported by the E2940A.	
B_E_ONLY_E2925A	This command is only supported by the E2925A.	
B_E_ONLY_E2925A_DEEP	This command is only supported by the E2925A deep trace.	
B_E_OVERFLOW	The value returned is larger than 0xFFFFFFFF. Actual value was <*value*>.	
B_E_PARALLEL_OPEN	Could not open BEST EPP port driver.	
B_E_PARAM	An internal parameter was out of range, unable to continue. Please check parameters of this call.	
B_E_PARAM_NOT_EXIST	In group <*group*> parameter <*parameter*> does not exist. Either the parameter or group index is out of range or the capability for the parameter is not enabled.	
B_E_PCI_NT_ONLY	Win32 PCI access is only supported under Windows NT.	
B_E_PCI_OPEN	Could not open BEST PCI driver.	
B_E_PROG_DEC_ENABLE	BEST IO programming decoder not enabled, or could not read from config space.	
B_E_RANGE	Parameter <*parameter*> has to be in a range from <*min*> through <*max*>. Its value is <*value*>.	
B_E_RS232_OPEN	Could not open RS232 port.	
B_E_SELFTEST_FAILED	Self test failed. Please contact support.	
B_E_SYNTAX	Syntax error.	
B_E_TEST_NO_DECODER	First decoder not enabled.	
B_E_UNDERFLOW	The value returned is less than 0. Actual value was <*value*>.	
B_E_UNKNOWN_ERR	Unknown error code <*errorcode*>.	
B_E_UNKNOWN_HARDWARE	The connected hardware on the <*port*> Port could not be identified.	

err/Error Code	Error String	Notes/Recommendations/Help
B_E_VALUE	Parameter *<parameter>* cannot take value *<value>* in this context.	
B_E_VERSION_MISMATCH	Expected version *<version1>* for *<programmable device>*. Current version is *<version2>*.	Try hardware update from graphical user interface.
B_E_WRONG_BUSADDR	Invalid PCI bus address specified in that address space—see PCI Spec.	
B_E_WRONG_PARAMETER	At least one parameter value is out of range.	
B_E_WRONG_PORT	This action is only applicable to the *<port1>* port. You are connected to the *<port2>* port.	

Firmware Errors

Error Code	Error string	Notes/ Recommendations/Help
B_EFW_ALIGN	Parameter *<parameter>* must be aligned to *<alignmt>*. Its value is *<value>*.	
B_EFW_ATTR_PROGMODE_MIXED	*<memory type> <page>* has to be programmed with *<progmode>*.	
B_EFW_BASEDEC_NOT_NORMAL	Decoder *<decoder1>* cannot be a base decoder for decoder *<decoder2>* because it is not switched to normal behavior.	Set *<decoder1>* to normal behav-ior.
B_EFW_BLOCKPAGE_CONCATENATED	Block page *<blockpage>* is concatenated.	
B_EFW_CARD_MEM_FULL	Not enough onboard memory.	
B_EFW_CMDBUSY	Cannot execute command *<command>*. Related resource is busy from port *<port>*.	
B_EFW_CMDLOCKED	Cannot execute command *<command>*. Related resource is locked by port *<port>*.	
B_EFW_CONFIG_CMD_SUBSET	Decoder *<decoder>* accepts only config commands in the command parameter. The current command mask is *<mask>*.	Select configuration commands for this decoder.
B_EFW_DATA_CHECKSUM	Data transmission error detected by checksum algorithm.	
B_EFW_DECODER_GRABBED	Decoder *<decoder>* is not accessible because its base address register is used as the upper half of a 64-bit decoder.	
B_EFW_DECODER_NO_DAC	Decoder *<decoder>* cannot decode DACs. Only odd numbered decoders have this capability.	
B_EFW_DEEPTRACE_FUNC	Deep Trace board may be defect!	
B_EFW_ERROR	Error during command transfer.	
B_EFW_EX_INIMODESET_FAILED	Programming failed, because *<reason>*.	The testcard was unable to switch to programming mode.

Error Code	Error string	Notes/ Recommendations/Help
B_EFW_FASTDECODER_COMMAND_ ERROR	Decoder *<decoder>* cannot be set to fast decoding speed because the command pattern *<cmdpatt>* is not supported at this speed. Please refer to the documentation for supported command sets.	Only the following decoders are capable of fast decoding: – Memory decoders, – I/O decoders, – Configuration (type 0) decoders. Use appropriate commands.
B_EFW_FASTDECODER_GRABBED	Decoder *<decoder1>* cannot be set to fast decoding speed because decoder *<decoder2>* is still set to fast speed.	
B_EFW_FUNC	Functional onboard error.	Runtime error on the hardware. Try again. If the problem persists, reboot the card.
B_EFW_ILLEGAL_PAGE_OFFSET	*<memory type> <page>* has only *<lines>* lines to address. Requested offset is *<offset>*.	
B_EFW_IMAGE_TOO_LARGE	The image is too large to be programmed into the ROM. Either the selected sector is too small or the RAM is too small to shadow the image.	Error while performing firmware update.
B_EFW_INVALID_ATTRPOINTER	Block *<block>* of page *<blockpage>* contains a pointer to attribute page *<attrpage>*, which is either empty, non-existent or concatenated.	
B_EFW_INVALID_PREP_ATTRPOINTER	Preparation register contains a pointer to attribute page *<attrpage>*, which is either empty, non-existent or concatenated.	
B_EFW_LIFELOCK_PREVENTION	The last command failed to prevent the system from a lifelock situation. One possible reason is a data memory up/download when the card is connected by PCI at the same time. Please refer to the manual for more information.	Use data memory accesses with less than 120 bytes.
B_EFW_MBLOCKCMD_DAC	Master cannot issue the DAC command directly. Use B_BLK_BUSDAC property to enable Dual Address Cycles.	
B_EFW_MEN_NOT_SET	Master cannot run, because the master enable bit is not set in config space.	
B_EFW_NO_CAPABIILITY	Your hardware is not capable of *<capability>* action.	
B_EFW_NO_EOP_FOUND	Could not find an EndOfPage (EOP) block in page *<blockpage>*.	
B_EFW_OK	No error.	
B_EFW_OVERLAY_CMD_SUBSET	The base decoder's *<decoder1>* decode commands have to be a superset of the overlaid decoder's *<decoder2>* range.	
B_EFW_OVERLAY_RANGE_SUBSET	The base decoder's *<decoder1>* decode range has to be a superset of the overlaid decoder's *<decoder2>* range.	
B_EFW_PAGE_CONCATENATED	*<memory type> <page>* is concatenated to its predecessor. Choose another page or initialize it first.	

Error Code	Error string	Notes/ Recommendations/Help
B_EFW_PAGE_EMPTY	*<memory type> <page>* is empty (initialized but not programmed yet).	
B_EFW_PARAM	An internal parameter was out of range, unable to continue. Please check parameters of this call.	
B_EFW_PARAM_NOT_EXIST	In group *<group>* parameter *<parameter>* does not exist. Either the parameter index is out of range or the capability for the parameter is not enabled.	
B_EFW_PATT_MTO_LISTSIGNAL	OR-operation on different list types in pattern.	Set the pattern string in such a way that signals of only one list type are OR-combined.
B_EFW_PATT_SYNTAX	Syntax error in *<pattern>*.	Syntax error in pattern string. The error can be found left of the marker (^).
B_EFW_PATT_UNDEF_TOKEN	Undefined token found in *<pattern>*.	The error can be found left of the marker (^).
B_EFW_PATT_UNKNOWN_ERROR	Error parsing *<pattern>*.	The error can be found left of the marker (^).
B_EFW_PCI_CLK_TOO_SLOW	PCI clock too slow or not running.	
B_EFW_PCI_NO_POWER	PCI bus is not powered.	
B_EFW_RANGE	Parameter *<parameter>* has to be in a range from *<min>* through *<max>*. Its value is *<value>*.	
B_EFW_SEQ_BUNDLE_SIGNAL	*<sequencer>*: Cannot output this value (too big) in Line: *<line>*.	
B_EFW_SEQ_TRAN_EXCLUSIVE	*<sequencer>*: "Next State" Transitions conditions of state *<line>* not exclusive.	
B_EFW_SEQ_SYNTAX_ERR	*<sequencer>*: Syntax Error in Line: *<line>*.	
B_EFW_SEQ_TOO_MANY_INPUTS	*<sequencer>*: Line *<line>*: Too many Inputs (max is *<maxinput>* with *<maxstates>* states).	
B_EFW_SEQ_TOO_MANY_STATES	*<sequencer>*: Line *<line>*: Too many states (max is *<maxstates>*).	
B_EFW_SEQ_UNKNOWN_OUTPUT	*<sequencer>*: Unknown output in Line: *<line>*	

Error Code	Error string	Notes/ Recommendations/Help
B_EFW_SEQ_UNKNOWN_TOKEN	*<sequencer>*: Unknown token in line: *<line>*	
B_EFW_SYNTAX	Syntax error.	
B_EFW_TARGET_BUSY	Programming failed, because the target is still active. Maybe the PCI clock is too slow or the target hangs.	The testcard was unable to switch to programming mode because the target is busy or the PCI clock is too slow. The PCI bus may be hanging. Resetting the statemachines may solve the problem.
B_EFW_VALUE	Parameter *<parameter>* cannot take value *<value>* in this context.	
B_EFW_WRONGPORT	Command *<command>* has to be issued from port *<port>*.	

b_exercisergenproptype

CAUTION

Changing the page size property of the attribute memory deletes all attribute pages!

exeprop (CLI Abbreviation)	value	Description
B_EGEN_ATTRPAGESIZE (attrpagesize)	**default:** 32	The attribute memory consists of 8 pages of **32 lines** each.
	4	The attribute memory consists of 64 pages of **4 lines** each. This provides more entry points.

b_mastergenproptype

mastergenprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_MGEN_ATTRMODE (attrmode)	The attribute mode property defines when the master attribute line pointer should be reset to the start of a master attribute page. **Note:** The block attribute B_BLK_CONTATTR must be set accordingly for this property to have an effect. *"BestMasterBlockPropSet" on page 125* sets this attribute.	
	default: B_ATTRMODE_BLOCK (block, 0)	Pointer is reset to start after each block.
	B_ATTRMODE_SEQUENTIAL (sequential, 1)	Pointer is not reset. The exerciser continues with the attributes of the next attribute memory line.
	B_ATTRMODE_PAGE (page, 2)	Pointer is reset to start after each block page.
B_MGEN_CACHELINESIZE (clinesize)	0 … 0xFF **default:** 0xFF	Sets the cacheline size in the register of the configuration space.
B_MGEN_DELAYCTR (delayctr)	32-bit value **default:** 0	Number of PCI clocks that are added to the inevitable delay between the occurrence of the conditional start pattern and the start of the master. The inevitable delay is caused by the testcard always inserting 4 clock cycles, and further delays if the bus is not granted to the master.
B_MGEN_INVDAT_ENABLE (invdataen)	0, 1 **default:** 1	When this bit is set, outgoing data bytes are inverted if they are masked by byte enables.
B_MGEN_LATCTR (latctr)	0 … 255 clocks **default:** 0	Sets the latency timer value of the testcard.
B_MGEN_LATMODE (latmode)	The latency mode property determines whether the master ignores or adheres to its latency timer. Ignoring the latency timer allows you to test beyond the PCI specification.	
	default: B_LATMODE_OFF (off)	Ignores the latency timer.
	B_LATMODE_ON (on)	Adheres to the latency timer.
B_MGEN_MASTERENABLE (men)	0, 1 **default:** 0	Sets the master enable bit in the command register of the testcard's configuration space.
B_MGEN_MWIENABLE (mwien)	0, 1 **default:** 0	Sets memory write and invalidate enable bit in the command register of the testcard's configuration space.
B_MGEN_REPEATMODE (repmode)	**default:** B_REPEATMODE_SINGLE (single, 1)	Sets the master to perform a master run only once.
	B_REPEATMODE_INFINITE (infinite, 0)	Sets the master to repeat a master run until *"BestMasterStop" on page 108* is called.

<table>
<tr><th>mastergenprop
(CLI Abbreviation)</th><th>value
(CLI Abbreviation)</th><th>Description</th></tr>
<tr><td rowspan="3">B_MGEN_RUNMODE
(runmode)</td><td colspan="2">The run mode property determines whether the master should start immediately, after a delay, or after a trigger event has occurred.

Note: This property has priority over the master block property for conditional start (see “b_blkproptype” on page 228, B_BLK_CONDSTART).</td></tr>
<tr><td>default:
B_RUNMODE_IMMEDIATE,
(immediate, 0)</td><td>Sets the master to start without waiting for the master start condition.</td></tr>
<tr><td>B_RUNMODE_WONDELAY,
(wondelay, 1)</td><td>Sets the master to wait until the master trigger pattern occurs, and the delay counter expires.</td></tr>
<tr><td>B_MGEN_TRYBACK_ENABLE
(backen)</td><td>0, 1
default: 0</td><td>Sets the Fast Back-to-Back bit in the command register of the configuration space.</td></tr>
<tr><td rowspan="3">B_MGEN_WARN64BIT
(warn64)</td><td colspan="2">If a 64-bit transfer moves an odd number of dwords to a memory block starting with an even address—and in the reverse case—a warning is issued because this can cause compare errors.

If the warning disturbs your test, use this property to suppress it.</td></tr>
<tr><td>default: 1</td><td></td></tr>
<tr><td>0</td><td></td></tr>
</table>

b_mattrgrouptype

NOTE The master attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

Master Attribute Group	group (CLI Abbreviation)	Master Attributes in the Group (CLI Abbreviation)
Address Phase Attributes	B_MATTR_GRP_MA0 (magrpa0)	B_M_DELAY (delay) B_M_DOLOOP* (loop)
	B_MATTR_GRP_MA1 (magrpa1)	B_M_DOLOOP* (loop) B_M_STEPS (steps)
	B_MATTR_GRP_MA2 (magrpa2)	B_M_LOCK* (lock) B_M_REQ64 (req64) B_M_RELREQ (rreq) B_M_DOLOOP* (loop)
	B_MATTR_GRP_MA3 (magrpa3)	B_M_DOLOOP* (loop) B_M_RESUMEDELAY (resdel)
	B_MATTR_GRP_MA4 (magrpa4)	B_M_APERR (aperr) B_M_AWRPAR (awp) B_M_AWRPAR64 (awp64) B_M_DACPERR (dacperr) B_M_DACWRPAR (dacwp) B_M_DACWRPAR64 (dacwp64) B_M_DOLOOP* (loop)
Data Phase Attributes	B_MATTR_GRP_MD0 (magrpd0)	B_M_DOLOOP* (loop) B_M_WAITS (w)
	B_MATTR_GRP_MD1 (magrpd1)	B_M_DOLOOP* (loop) B_M_DPERR (dperr) B_M_DSERR (dserr) B_M_DRELREQ* (drreq)
	B_MATTR_GRP_MD2 (magrpd2)	B_M_DOLOOP* (loop) B_M_DWRPAR (dwp) B_M_DWRPAR64 (dwp64) B_M_MARKER* (mark)
Control Attributes	B_MATTR_GRP_ML (magrpl)	B_M_DOLOOP* (loop) B_M_LAST (last) B_M_REPEAT* (repeat)

b_mattrproptype

The master attribute properties of type b_mattrproptype are divided into the following sections:

- *"Address Phase Attributes (Master)" on page 251*,
- *"Data Phase Attributes (Master)" on page 254*,
- *"Control Attributes (Master)" on page 256*.

Address Phase Attributes (Master)

NOTE The master attribute marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

mattrprop (CLI Abbreviation)	**value**	**Description**
B_M_APERR (aperr)	Asserts a system error (SERR#) in the address phase (SERR# is used to signal address parity errors). This property also sets the SERR flag in the configuration space and in the status register of the testcard. System errors must be enabled in the configuration space for this property to have an effect.	
	default: 0	No error is signaled.
	1	Error is signaled.
B_M_AWRPAR (awp)	A wrong parity is set one clock after the address phase.	
	default: 0	Parity remains as it is.
	1	Parity is inverted.
B_M_AWRPAR64 (awp64)	A wrong parity (PAR64) is set one clock after the address phase or in the first half of a dual address cycle.	
	default: 0	Parity remains as it is.
	1	Parity is inverted.
B_M_DACPERR (dacperr)	Asserts a system error (SERR#) in the second cycle of a dual address cycle (SERR# is used for address parity errors). This property also sets the SERR flag in the configuration space and in the status register of the testcard. System errors must be enabled in the configuration space for this property to have an effect.	
	default: 0	No error is signaled.
	1	Error is signaled.
B_M_DACWRPAR (dacwp)	A wrong parity signaled in the second cycle of a dual address cycle.	
	default: 0	Parity remains as it is.
	1	Parity is inverted.

mattrprop (CLI Abbreviation)	**value**	**Description**
B_M_DACWRPAR64 (dacwp64)	A wrong parity (PAR64) signaled in the second cycle of a dual address cycle.	
	default: 0	Parity remains as it is.
	1	Parity is inverted.
B_M_DELAY (delay)	0 … 2^{21} **default:** 0	Number of clocks a master transaction is delayed before its start. The transactions must reside in the same block, otherwise an additional gap of 15 clocks is inserted. The master stays idle only if B_M_TRYBACK (Fast Back-to-Back) is not activated. **Note:** The master must also stay idle until the arbiter grants the bus access to the master. This can result in additional delay clocks.
B_M_LOCK* (lock)	To achieve exclusive ownership of a target during an access, the master can lock the target with the LOCK# signal. This property controls whether or not an access is exclusive and should remain unchanged during the entire transaction, unless "hide_lock" is used. Exclusive access can only be established with a "read" command.	
	B_LOCK_LOCK (lock, 1)	The master tries to achieve exclusive access to the target.
	B_LOCK_HIDELOCK (hidelock, 3)	The exerciser will not release LOCK# after the address phase. This is useful to test the lock protocol of the target with one master only. The exerciser can pretend to be another master trying to access the locked target. **Note:** This value is used only once. After a retry, the next attribute line is used.
	B_LOCK_UNCHANGE (unchange, 0)	LOCK# state remains unchanged.
	default: B_LOCK_UNLOCK (unlock, 2)	Releases LOCK# at the end of this transaction. The master will not try to achieve exclusive access to the target.
B_M_RELREQ (rreq)	Allows you to deassert REQ# earlier than usual (usual means: one clock after the bus enters the idle state after a transfer).	
	B_RELREQ_ON (rreqon, 0)	REQ# is released directly after FRAME# is asserted.
	1 … 14	Number of cycles after which REQ# is released after assertion of FRAME#.
	default: B_RELREQ_OFF (rreqoff, 15)	The master releases REQ# as usual.
B_M_REQ64 (req64)	The exerciser tries a 64-bit transfer and asserts REQ64# together with FRAME#. The address must be aligned to a qword boundary, otherwise REQ64# is not asserted. If an intended 64-bit data transfer has been denied once within a block, the exerciser assumes that this address range is not capable of handling 64-bit data accesses and will not try to transfer 64-bit data in this block again.	
	default: 0	No 64-bit access is tried.
	1	64-bit access is tried.

mattrprop (CLI Abbreviation)	**value**	**Description**
B_M_RESUMEDELAY (resume)	2 … 127 **default:** 10	Number of clocks after which the master resumes after a target termination. The PCI Specification requires the master to deassert REQ# after a target termination. This parameter specifies the number of clock cycles after which the master reasserts REQ#. The purpose of this parameter is to give other masters a programmable chance to obtain the bus. **Note:** If the exerciser has already been granted access to the bus (parking master), the master resumes after 2 clocks.
B_M_STEPS (steps)	0 … 15 **default:** 0	Number of additional clocks during an address phase. They are added between assertion of GNT# and assertion of FRAME#.
B_M_TRYBACK (back)	The exerciser can try a Fast Back-to-Back cycle if this is allowed by PCI specification in the current situation, and if this is enabled in the configuration space. **Notes:** Only transactions within one block transfer can be executed with Fast Back-to-Back. The number of additional address steps (B_M_STEPS) must be 0.	
	default: 0	Does not try Fast Back-to-Back cycle.
	1	Tries Fast Back-to-Back cycle.

Data Phase Attributes (Master)

NOTE The master attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

<table>
<tr><th>mattrprop
(CLI Abbreviation)</th><th>value</th><th>Description</th></tr>
<tr><td rowspan="3">B_M_DPERR
(dperr)</td><td colspan="2">Asserts a parity error (PERR#) two clocks after the read data transfer and sets the PERR flag in the configuration space and in the status register of the testcard. Parity errors must be enabled in the configuration space for this property to have an effect.

Note: This attribute is ignored in write transfers. It is also ignored in read transfers when data comparison is enabled (B_BLK_COMPFLAG is 1 and B_BLK_BUSCMD is a “read” command; see “b_mattrproptype” on page 251).</td></tr>
<tr><td>default: 0</td><td>No error is signaled.</td></tr>
<tr><td>1</td><td>Error is signaled.</td></tr>
<tr><td rowspan="4">B_M_DRELREQ*
(drreq)</td><td colspan="2">Allows you to deassert REQ# earlier than usual (usual means: one clock after the bus enters the idle state after the transfer).

Note: This entry is ignored if the master has already been instructed to release REQ# (with B_M_RELREQ in the address phase, or with B_M_DRELREQ in a previous data phase within the same burst).</td></tr>
<tr><td>B_DRELREQ_ON
(drreqon, 0)</td><td>REQ# is released immediately after the start of the data phase.</td></tr>
<tr><td>1 … 30</td><td>Number of cycles after the start of the data phase.</td></tr>
<tr><td>default:
B_DRELREQ_OFF (drreqoff, 31)</td><td>The master releases REQ# as usual.</td></tr>
<tr><td rowspan="3">B_M_DSERR
(dserr)</td><td colspan="2">Asserts a system error (SERR#) in the data phase along with IRDY# and sets the SERR flag in the configuration space and in the status register of the testcard.

System errors must be enabled in the configuration space for this property to have an effect.</td></tr>
<tr><td>default: 0</td><td>No error is signaled.</td></tr>
<tr><td>1</td><td>Error is signaled.</td></tr>
<tr><td rowspan="3">B_M_DWRPAR
(dwp)</td><td colspan="2">A wrong parity (PAR) is set one clock after a write data transfer. This attribute is ignored in read transfers.</td></tr>
<tr><td>default: 0</td><td>Parity remains as it is.</td></tr>
<tr><td>1</td><td>Parity is inverted.</td></tr>
</table>

mattrprop (CLI Abbreviation)	value	Description
B_M_DWRPAR64 (dwp64)	A wrong parity (PAR64) is set one clock after a write data transfer. This attribute is ignored in read transfers.	
	default: 0	Parity remains as it is.
	1	Parity is inverted.
B_M_MARKER* (marker)	0 … 15 **default:** 0	Number issued during the address and data phase. It can be observed by pattern terms, for example, for synchronization. By convention, the value “0” is used to indicate “nothing special”.
B_M_WAITS (w)	Number of waits. This is controlled by the IRDY# behavior per data phase. **Note for 64-bit data transfers**: If the target does not accept an intended 64-bit data transfer, then the master asserts IRDY# at the earliest two clocks after the target has asserted DEVSEL#. In practice, this means no performance degradation, because single 64-bit data accesses are not recommended by the PCI Specification, because the master has to insert initial waits until the target accepts 64-bit data accesses.	
	B_M_WAITS_HANG (whang, –1)	Simulates a hanging master (IRDY# is not asserted).
	0 … 30 **default:** 0	Number of waits inserted.

Control Attributes (Master)

NOTE The master attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

mattrprop (CLI Abbreviation)	value	Description
B_M_DOLOOP* (loop)	Loop bit. This bit indicates the last line in the attribute page. The next line executed after this one is again the first line of the same page. Used with *"BestMasterAttrGroupLineProg" on page 113*, the loop bit is set for the corresponding master attribute *group*. Used with *"BestMasterAttrLineProg" on page 115*, the *general loop bit* for the complete page is set (previously set group loop bits for this page are overwritten).	
	default: 0	Continues with the next attribute line.
	1	Continues with the first line of the same page.
B_M_LAST (last)	1 … 2^{32} **default:** 0	Indicates the last phase of a burst. **Note:** When used in combination with the block command B_CMD_MEM_WRITEINVALIDATE, illegal burstlengths can be generated.
B_M_REPEAT* (repeat)	One attribute line can be performed repeatedly.	
	0	Line used for all transfers.
	1 … 2^{32} **default:** 1	Number of repetitions.

b_obsruletype

obsrule (CLI Abbreviation)	obsrule (CLI Abbreviation)	obsrule (CLI Abbreviation)
B_R_FRAME_0 (frame_0)	B_R_LOCK_1 (lock_1)	B_R_SEM_0 (sem_0)
B_R_FRAME_1 (frame_1)	B_R_LOCK_2 (lock_2)	B_R_SEM_1 (sem_1)
B_R_IRDY_0 (irdy_0)	B_R_CACHE_0 (cache_0)	B_R_SEM_2 (sem_2)
B_R_IRDY_1 (irdy_1)	B_R_CACHE_1 (cache_1)	B_R_SEM_3 (sem_3)
B_R_IRDY_2 (irdy_2)	B_R_PARITY_0 (par_0)	B_R_SEM_4 (sem_4)
B_R_IRDY_3 (irdy_3)	B_R_PARITY_1 (par_1)	B_R_SEM_5 (sem_5)
B_R_IRDY_4 (irdy_4)	B_R_PARITY_2 (par_2)	B_R_SEM_6 (sem_6)
B_R_DEVSEL_0 (devsel_0)	B_R_W64_0 (w64_0)	B_R_SEM_7 (sem_7)
B_R_DEVSEL_1 (devsel_1)	B_R_W64_1 (w64_1)	B_R_SEM_8 (sem_8)
B_R_DEVSEL_2 (devsel_2)	B_R_W64_2 (w64_2)	B_R_SEM_9 (sem_9)
B_R_DEVSEL_3 (devsel_3)	B_R_W64_3 (w64_3)	B_R_SEM_10 (sem_10)
B_R_TRDY_0 (trdy_0)	B_R_ARB_0 (arb_0)	B_R_SEM_11 (sem_11)
B_R_TRDY_1 (trdy_1)	B_R_ARB_1 (arb_1)	B_R_SEM_12 (sem_12)
B_R_TRDY_2 (trdy_2)	B_R_ARB_2 (arb_2)	B_R_SEM_13 (sem_13)
B_R_STOP_0 (stop_0)	B_R_PARITY_3 (par_3)	B_R_LAT_0 (lat_0)
B_R_STOP_1 (stop_1)	B_R_PARITY_4 (par_4)	B_R_LAT_1 (lat_1)
B_R_STOP_2 (stop_2)	B_R_PARITY_5 (par_5)	B_R_IRDY_5 (irdy_5)
B_R_LOCK_0 (lock_0)	B_R_PARITY_6 (par_6)	

For detailed rule descriptions, see *Agilent E2940A PCI Analyzer User's Guide*.

b_obsstatustype

The following identifiers define the observer register to be read with *"BestObsStatusGet" on page 51*.

obsstatus (CLI Abbreviation)	Description
B_OBS_FIRSTERR (firsterr)	First Error Register (bit 0 … 31). The returned values indicate the first error that has occurred after observer startup.
B_OBS_FIRSTERR2 (firsterr2)	First Error Register (bit 32 … 63).
B_OBS_ACCUERR (accuerr)	Accumulated Error Register (bit 0 … 31). The returned value indicates all protocol errors that have occurred since startup of the observer.
B_OBS_ACCUERR2 (accuerr2)	Accumulated Error Register (bit 32 … 63).
B_OBS_OBSSTAT (obsstat)	Observer Status Register. The returned value indicates the observer status, see table below.

Error Register

Each bit in the error registers represents one rule. If the rule has been masked, the bit value is undefined. For a list of rule identifiers refer to *"b_obsruletype" on page 257*.

Observer Status Register

The observer status register indicates the status of the observer.

Bits	Meaning
0	1= observer is currently in run mode
1	1= observer out of sync
2	1= protocol error detected
[31::3]	not used, returns 0

b_perfgenproptype

perfgenprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_PERFGEN_CAMODE (mode)	**default:** B_CAMODE_INCR1 (incr1, 0)	Counter A is incremented by 1 after each count enable.
	B_CAMODE_INCRBYTEN (incrbyten, 1)	If the count enable is active, counter A is incremented by the number of byte enables set (C/BE3[3:0]) in order to count data.
B_PERFGEN_CTRC_PREL (cprel)	0 … 2^{32} – 1 **default:** 0	Preload value for feedback counter C.

b_perfseqtrancondproptype

perfseqtrancondprop (CLI Abbreviation)	Default values of condition (CLI Abbreviation)	Description
B_PERFSEQ_CA_EN caen)	"0" (false)	Condition to increment counter A (nominator counter).
B_PERFSEQ_CB_EN (cben)	"0" (false)	Condition to increment counter B (denominator counter).
B_PERFSEQ_CE (ce)	"0"(false)	Condition to decrement feedback counter C.
B_PERFSEQ_CLOAD (cload)	"0" (false)	Condition to preload feedback counter C.
B_PERFSEQ_XCOND (x)	"1" (true)	Transition condition for a performance sequencer to move from one state to the next.

b_perfseqtranproptype

perfseqtranprop (CLI Abbreviation)	**value**	**Description**
B_PERFSEQ_STATE (state)	0 … 63 **default:** 0	State identifier. The sequencer starts at state 0.
B_PERFSEQ_NEXTSTATE (nextstate)	0 … 63 **default:** 0	Identifier for the state after the transition condition has been met.

b_porttype

port	portnum	Description
B_PORT_FASTHIF	0 … n	Specifies a connection via fast host interface card. The first fast host interface card to be found is assigned to "0", the second to "1", and so forth. The order in which the cards are found is not predictable. After the BestOpen call, use *"BestPing" on page 22* and watch the LEDs to see which card is connected to the session.
B_PORT_OFFLINE	Result of: *assumed* capabilities OR *assumed hardware* Values for *assumed hardware* are: B_HW_E2925B, B_HW_E2925B_DEEP, B_HW_E2926A, B_HW_E2926A_DEEP, B_HW_E2926B, B_HW_E2926B_DEEP, B_HW_E2927A, B_HW_E2927A_DEEP, B_HW_E2928A, B_HW_E2928A_DEEP, B_HW_E2940A, B_HW_E2940A_DEEP	Specifies the Offline/Demo Mode. The Offline/Demo Mode allows you to call and try out functions during test development without hardware. In Offline/Demo Mode, the calls still perform most of the runtime range checking. The port number must be the result of the logical OR-combination of the assumed hardware and the assumed activated capability codes. The values for activated capabilities are returned by the BestCapabilityCheck call, see *"Capability Code Values" on page 25*. **Example:** (B_HW_E2926A_DEEP | B_CAPABILITY_64_BIT)
B_PORT_PCI_CONF	32 Bit	Device number of the testcard, as used by PCI BIOS and host bridge. This number may be requested using the function *"BestDevIdentifierGet" on page 19* in a system with PCI BIOS. If the system does not have a PCI BIOS, then you must enter a specific system identifier to identify the testcard (see *"Device Identifier Format" on page 20*).
B_PORT_RS232	B_PORT_COM1, B_PORT_COM2, B_PORT_COM3, B_PORT_COM4	Specifies a serial port.
B_PORT_CURRENT		Specifies the current session.

b_puproptype

<table>
<tr><th>pu_prop (CLI Abbreviation)</th><th>value (CLI Abbreviation)</th><th>Description</th></tr>
<tr><td rowspan="2">B_PU_MASTERRRUNMODE (mrunmode)</td><td>0 … 15</td><td>Provides the number of the block page the master executes after power up.</td></tr>
<tr><td>default: M_RUN_OFF</td><td>The master does not start automatically after power up.</td></tr>
<tr><td rowspan="2">B_PU_TRCRUNMODE (trcrunmode)</td><td>default: 0</td><td>The trace memory is started not automatically after power up.</td></tr>
<tr><td>1</td><td>The trace memory runs after power up.</td></tr>
<tr><td rowspan="3">B_PU_CONFRESTORE (confrestore)</td><td colspan="2">Behavior of the testcard’s configuration space after power up or execution of “BestAllPropStore” on page 34 (this affects primarily the base address registers).</td></tr>
<tr><td>default: 0</td><td>Read/write bits of the configuration space is set to 0.</td></tr>
<tr><td>1</td><td>Read/write bits of the configuration space are left as they are.</td></tr>
<tr><td rowspan="3">B_PU_SSTRUNMODE (sstrunmode)</td><td colspan="2">Used by the Agilent E2974A system stress test software. Do not directly set it to 1, but it may be set to 0 to disable a test which has already been set up using the stress test software.</td></tr>
<tr><td>default: 0</td><td>No subsystem test is run at power up.</td></tr>
<tr><td>1</td><td>An Agilent E2974A test is run at power up.</td></tr>
</table>

b_resourcetype

resource (CLI Abbreviation)	Description
B_RESLOCK_EXERCISER (exe)	Exerciser
B_RESLOCK_MAILBOX (mailbox)	Mailbox registers
B_RESLOCK_OBSERVER (obs)	Observer
B_RESLOCK_PATT_TERM0 (pt0) … B_RESLOCK_PATT_TERM23 (pt23)	Pattern terms
B_RESLOCK_PERFORMANCE (perfor)	Performance measures
B_RESLOCK_TRACEMEM (trc)	Trace memory

b_signaltype (for Timing Check)

The following table shows the signals checked by the PCI Analyzer's timing checker to detect timing violations. The remaining signals are not checked due to different timing behavior.

signal	Timing Checker Syntax	Description
B_SIG_ACK64	ACK64	Target acknowledgment of a 64-bit transfer.
B_SIG_AD32	AD32	Lower 32 address and data bus lines.
B_SIG_AD64	AD64	Upper 32 address and data bus lines.
B_SIG_CBE3_0	CBE3_0	Lower 4 PCI command and byte enables sideband signals.
B_SIG_CBE7_4	CBE7_4	Upper 4 PCI command and byte enables sideband signals.
B_SIG_DEVSEL	DEVSEL	Device Select
B_SIG_FRAME	FRAME	Cycle Frame. Indicates start and duration of a transaction.
B_SIG_IDSEL	IDSEL	Own Initialization Device Select
B_SIG_IRDY	IRDY	Initiator (= Master) Ready
B_SIG_LOCK	LOCK	Lock
B_SIG_PAR	PAR	Parity Bit
B_SIG_PAR64	PAR64	Parity Bit for AD[63::32]
B_SIG_PERR	PERR	Parity Error
B_SIG_REQ64	REQ64	Master request for a 64-bit transfer.
B_SIG_SBO	SBO	Snoop Back Off
B_SIG_SDONE	SDONE	Snoop Done
B_SIG_SERR	SERR	System Error
B_SIG_STOP	STOP	Stop
B_SIG_TRDY	TRDY	Target Ready
B_SIG_trigger0 ... B_SIG_trigger11	trigger0 ... trigger11	External trigger lines of the testcard.

b_signaltype (List of Signals)

The topics included in this section list bus signals, bus states, etc. that are stored in the trace memory and/or can be used in pattern terms. Any restrictions in the use of signals are stated explicitly.

The signals are grouped as follows:

- *"Bus Signals" on page 267*

  These signals are directly taken from the PCI bus lines.

- *"Bus States" on page 268*

  Bus states are internally generated by the testcard to provide easy trigger setup for the multiplexed PCI bus.

- *"Transaction Attributes" on page 271*

  These signals can be used for triggering on the occurrence of certain transfer attributes, for example, commands

- *"Markers" on page 272*

  Markers are issued by the master or target on the testcard.

- *"Exerciser Signals" on page 273*

  Exerciser signals are derived from the states of master and target.

- *"Internal Counters" on page 273*

  Values of internal counters (stored in trace memory).

- *"Checks" on page 273*

  These signals can be used for triggering on the basis of results of data compare and detected timing or protocol errors.

- *"Gap Information" on page 274* (if gap mode is set to "performance")

  Gap information stored in trace memory instead of samples due to selective recording by the storage qualification.

Signal Types

The following table shows the types of signals, their capabilities and where they can be applied. The “type” column of the tables in the following topics shows the type of each signal.

Criteria Type	Capability	Application
10X	For the individual bits, you can determine whether they are 1, or 0, or “don’t care” (“don’t care” = X).	Single-bit signals, for example, FRAME# or IRDY#.
10X vector	For multiple bit signals, you can determine for each bit, whether it is 1, or 0, or “don’t care” (“don’t care” = X).	Bitwise signals, for example, byte enables.
10 vector	For multiple bit signals, you can determine for each bit, whether it is 1, or 0.	Bitwise signals, for example, some bus addresses.
list	For multiple bit signals that are encoded in multiple bit values, you can determine these values. The single bits have no meaning.	For example, commands. The elements of each list are numbered, for example, command “I/O Read” equals 2. Only the numeric values may be used.
range	For multiple bit signals of this type, you can determine whether their value is within a certain range. The following constraints can be set: <, >, <>, >=, <=, ==, don’t care.	Buses that represent a value, for example, address or counters.

Bus Signals

These signals are directly taken from the PCI bus lines.

signal	Pattern and Trace Memory Syntax	Type	Description
B_SIG_ACK64	ACK64	10X	64-Bit Acknowledge
B_SIG_AD32	AD32	10X vector	Lower 32 address and data bus lines.
B_SIG_AD64	AD64	10X vector	Upper 32 address and data bus lines.
–	addr_phase	10X	Indicates an address phase.
B_SIG_CBE3_0	CBE3_0	10X vector	Lower 4 PCI command and byte enables sideband signals.
B_SIG_CBE7_4	CBE7_4	10X vector	Upper 4 PCI command and byte enables sideband signals.
B_SIG_DEVSEL	DEVSEL	10X	Device Select
B_SIG_FRAME	FRAME	10X	Cycle Frame. Indicates whether a transaction is running.
B_SIG_GNT	GNT	10X	Own Grant
B_SIG_IDSEL	IDSEL	10X	Own Initialization Device Select
B_SIG_INTA … B_SIG_ INTD	INTA … INTD	10X	Interrupt A … D
B_SIG_IRDY	IRDY	10X	Initiator (= Master) Ready
B_SIG_LOCK	LOCK	10X	Lock
B_SIG_PAR	PAR	10X	Parity Bit
B_SIG_PAR64	PAR64	10X	Parity Bit for upper 32 address and data bus lines.
B_SIG_PERR	PERR	10X	Parity Error
B_SIG_REQ	REQ	10X	Own Bus Request
B_SIG_REQ64	REQ64	10X	64-bit Request
B_SIG_RESET	RESET	10X	Reset
B_SIG_SBO	SBO	10X	Snoop Back Off
B_SIG_SDONE	SDONE	10X	Snoop Done
B_SIG_SERR	SERR	10X	System Error
B_SIG_STOP	STOP	10X	Stop
B_SIG_TRDY	TRDY	10X	Target Ready
B_SIG_trigger0 … B_SIG_trigger11	trigger0 … trigger11	10X	External trigger lines of the testcard.

Bus States

Bus states are detected and made available by the analyzer when observing protocol rules.

<table>
<tr><th>signal</th><th>Pattern and Trace Memory Syntax</th><th>Type</th><th colspan="2">Meaning and Values</th></tr>
<tr><td rowspan="9">B_SIG_b_state</td><td rowspan="9">b_state</td><td rowspan="9">list</td><td colspan="2">The current state of the bus as detected by the observer.

Note: To identify the state use the **numeric values**.</td></tr>
<tr><td>unsync = 0</td><td>After reset, this state is active until an idle state (FRAME# & IRDY#) is detected.

After a protocol violation, this state is used if no other state can be assumed.</td></tr>
<tr><td>idle = 1</td><td>The bus is idle (FRAME# & IRDY#).</td></tr>
<tr><td>dac1 = 2</td><td>The first half of a dual address cycle.</td></tr>
<tr><td>addr = 3</td><td>Single address phase.</td></tr>
<tr><td>dac2 = 4</td><td>The second half of a dual address cycle.</td></tr>
<tr><td>decoding = 5</td><td>The address phase has been completed, however, a target has not yet claimed the access.</td></tr>
<tr><td>wait = 6</td><td>A target has accepted the transfer (asserted DEVSEL#). The target, the master, or both are inserting wait cycles.

If IRDY# is deasserted, the waits are inserted by the master, if TRDY# is deasserted, they are inserted by the target.</td></tr>
<tr><td>transfer = 7</td><td>Currently a data transfer is taking place (both IRDY# and TRDY# are asserted).</td></tr>
</table>

<table>
<tr><th>signal</th><th>Pattern and Trace Memory Syntax</th><th>Type</th><th colspan="2">Meaning and Values</th></tr>
<tr><td rowspan="7">–</td><td rowspan="7">decode</td><td rowspan="7">list</td><td colspan="2">Decode speed of the current transaction.

Once a decode speed has been determined, the information stays valid during the entire transaction. Master aborts are detected by the termination state.</td></tr>
<tr><td>dont_know = 0</td><td>A transaction has not been started or a started transaction has not yet been claimed by a target. This state is left when b_state is “wait” or “transfer”.</td></tr>
<tr><td>fast = 1</td><td>The current transaction has been decoded with fast decode speed.</td></tr>
<tr><td>medium = 2</td><td>The current transaction has been decoded with medium decode speed.</td></tr>
<tr><td>slow = 3</td><td>The current transaction has been decoded with slow decode speed.</td></tr>
<tr><td>subtractive = 4</td><td>The current transaction has been decoded with subtractive decode speed or one clock slower than “slow”.</td></tr>
<tr><td>too_slow = 5</td><td>The current transaction has been decoded slower than a subtractive decoder decodes.</td></tr>
<tr><td rowspan="6">B_SIG_burst_order</td><td rowspan="6">burst_order</td><td rowspan="6">list</td><td colspan="2">Indicates at which position of a burst a data phase was started by the master, independent of whether it is actually terminated earlier.

This information is only valid in the last clock of a data phase and can be used to count types of data phases with respect to its position within the burst.

Note: A target termination with no data transfer and target limited waits is counted as one data phase.</td></tr>
<tr><td>invalid = 0</td><td>This is not the last clock of a data phase.</td></tr>
<tr><td>single = 1</td><td>This data phase belongs to a burst of length one.</td></tr>
<tr><td>first = 2</td><td>This is the first data phase in a burst of a minimum of two data phases.</td></tr>
<tr><td>middle = 3</td><td>This is a data phase between first and last in a burst of minimum three data phases.</td></tr>
<tr><td>last = 4</td><td>This is the last data phase in a burst of a minimum of two data phases.</td></tr>
</table>

<table>
<tr><th>signal</th><th>Pattern and Trace Memory Syntax</th><th>Type</th><th colspan="2">Meaning and Values</th></tr>
<tr><td rowspan="9">B_SIG_term</td><td rowspan="9">term</td><td rowspan="9">list</td><td colspan="2">Termination of a data phase.

It can be used to count the occurrence of certain terminations. A target termination with no data transfer and target limited waits is counted as one data phase.

This information is only valid in the last data phase of a transaction. Therefore, the information is valid only if the signal burst is valid simultaneously (any burst value except “invalid”).</td></tr>
<tr><td>no_term = 0</td><td>This data phase is not the end of a transaction.</td></tr>
<tr><td>m_abort = 1</td><td>The transaction is terminated by the master with the master abort protocol.</td></tr>
<tr><td>m_complete = 2</td><td>The master has concluded its intended transaction.

This may also be because its latency timer has expired and it has to conclude the transaction.</td></tr>
<tr><td>t_disconnectA = 3</td><td>This transaction is terminated by the target. Data has been transferred successfully.</td></tr>
<tr><td>t_disconnectB = 4</td><td>This transaction is terminated by the target. Data has been transferred successfully.</td></tr>
<tr><td>t_disconnect1/t_retry1 = 5</td><td>This transaction is terminated by the target without any data transfers.

When burst is in the “single” or “first” state, this is a “retry”, otherwise a “disconnect” (formerly “disconnect-C”).</td></tr>
<tr><td>t_disconnect2 / t_retry2 = 6</td><td>This transaction is terminated by the target without any data transfers.

When “burst” is in the “single” or “first” state, this is a “retry”, otherwise a “disconnect” (formerly “disconnect-C”).</td></tr>
<tr><td>t_abort = 7</td><td>This transaction is terminated by the target using a target abort protocol.</td></tr>
<tr><td>–</td><td>berr</td><td>10X</td><td colspan="2">Protocol violation detected.</td></tr>
</table>

Transaction Attributes

Protocol attributes accompanying the transaction.

<table>
<tr><th>signal</th><th>Pattern and Trace Memory Syntax</th><th>Type</th><th colspan="2">Meaning and Values</th></tr>
<tr><td>–</td><td>xact_fb2b</td><td>10X</td><td colspan="2">Fast Back-to-Back.
This signal indicates whether the current transaction has started back-to-back to the previous one. Otherwise, the current transaction has started after at least one idle state.
The signal is valid between the address phase and the end of the transaction (while FRAME# or IRDY# is low).</td></tr>
<tr><td>B_SIG_xact_dac</td><td>xact_dac</td><td>10X</td><td colspan="2">Dual Address Cycle.
This signal indicates that a current transaction addresses to a 64-bit address space. Otherwise, it addresses a 32-bit space.
The signal is valid between the address phase and the end of the transaction (while FRAME# or IRDY# is low).</td></tr>
<tr><td rowspan="17">B_SIG_xact_cmd</td><td rowspan="17">xact_cmd</td><td rowspan="17">list</td><td colspan="2">Command used for the current transaction.
This information is valid between the address phase and the end of the transaction (while FRAME# or IRDY# is low).
In a dual address cycle it shows “DAC” in the first address cycle, and the bus command used in the second address cycle.</td></tr>
<tr><td>0</td><td>Interrupt Acknowledge</td></tr>
<tr><td>1</td><td>Special Cycle</td></tr>
<tr><td>2</td><td>I/O Read</td></tr>
<tr><td>3</td><td>I/O Write</td></tr>
<tr><td>4</td><td>Reserved</td></tr>
<tr><td>5</td><td>Reserved</td></tr>
<tr><td>6</td><td>Memory Read</td></tr>
<tr><td>7</td><td>Memory Write</td></tr>
<tr><td>8</td><td>Reserved</td></tr>
<tr><td>9</td><td>Reserved</td></tr>
<tr><td>A</td><td>Configuration Read</td></tr>
<tr><td>B</td><td>Configuration Write</td></tr>
<tr><td>C</td><td>Memory Read Multiple</td></tr>
<tr><td>D</td><td>Dual Address Cycle</td></tr>
<tr><td>E</td><td>Memory Read Line</td></tr>
<tr><td>F</td><td>Memory Write & Invalidate</td></tr>
</table>

<table>
<tr><th>signal</th><th>Pattern and Trace Memory Syntax</th><th>Type</th><th colspan="2">Meaning and Values</th></tr>
<tr><td rowspan="4">B_SIG_xact_burst</td><td rowspan="4">xact_burst</td><td rowspan="4">list</td><td colspan="2">Indicates the dword ordering of a burst to cacheable memory space for the current transaction. For details, refer to PCI Specification, “Addressing”.

During accesses to memory space, this signal equals AD[1::0] in the address phase, otherwise it is set to “linear_incr”.

This information is valid between the address phase and the end of the transaction (while FRAME# or IRDY# low).</td></tr>
<tr><td>linear_incr = 0</td><td>linear increment</td></tr>
<tr><td>wrap_around = 2</td><td>cacheline wrap around</td></tr>
<tr><td>reserved_01 = 1
reserved_11 = 3</td><td>reserved encodings</td></tr>
<tr><td>B_SIG_xact_req64</td><td>xact_req64</td><td>10X</td><td colspan="2">This signal indicates whether the master requested a 64-bit data transfer in the address phase.

This information is valid between the address phase and the end of the transaction (while FRAME# or IRDY# low).</td></tr>
<tr><td>–</td><td>xact_lock</td><td>10X</td><td colspan="2">This signal indicates whether the current transaction is an exclusive access.

This information is valid between the address phase and the end of the transaction (while FRAME# or IRDY# low).</td></tr>
<tr><td>–</td><td>xact_trig0
...
xact_trig11</td><td>10X</td><td colspan="2">Shows the states of trigger 0 to 11 one clock before the most recent address phase.

If the trigger inputs are connected to the GNT# signals of other devices, this signal can be used to identify the master of the current transaction.

This information is valid during the whole transaction (while FRAME# or IRDY# are low).</td></tr>
</table>

Markers

The markers can be used in standard pattern terms only.

signal	Pattern Syntax	Type	Description
–	t_marker[3::0]	10X	Marker generated by the testcard’s Exerciser target. This signal can be used to relate trigger outputs to target transactions. Aligned to data.
–	m_marker[3::0]	10X	Marker generated by the testcard’s Exerciser master. This signal can be used to relate trigger outputs to master transactions. Aligned to data.

Exerciser Signals

Exerciser signals are derived from status of master and target.

signal	Pattern and Trace Memory Syntax	Type	Meaning
B_SIG_m_act	m_act	10X	Master of the testcard is active.
B_SIG_m_lock	m_lock	10X	Master of the testcard has established a lock.
B_SIG_t_act	t_act	10X	Target of the testcard is active.
–	master_done	10X	The intended master block page has been completed. This signal can be used to inform the sequencer about completion. A pattern output using this signal will be true 8 clocks after the master has completed the last data phase.
B_SIG_t_lock	t_lock	10X	Target of the testcard is locked.

Internal Counters

Values of internal counters.

Counter	Pattern and Trace Memory Syntax	Type	Meaning
B_SIG_block_xfer	block_xfer	number	8 bits. Identifies the currently executed block number.
B_SIG_attr_ctr	attr_ctr	number	8 bits. Attribute counter. This is the *master* attribute counter whenever the master is involved in the stored transaction. Otherwise, it is the *target* attribute counter.

Checks

Results of data compare and detected protocol errors.

signal	Pattern and Trace Memory Syntax	Type	Meaning
B_SIG_prot_rule	prot_rule	10X	Protocol violation has occurred. The signal is aligned to bus signals.
B_SIG_chck_data	chck_data	10X	Data discrepancy (miscompare) has occurred. The signal is aligned to a data transfer.
–	timing_err	10X	Timing error has occurred.

Gap Information

Gap information stored instead of samples due to storage qualification.

signal	Pattern and Trace Memory Syntax	Type	Meaning
B_SIG_gap	gap	10X	Gap flag, indicating that a gap precedes this sample due to storage qualification. The information stored in this memory line depends on the gap mode setting. If gap mode is set to “performance”, information is as described in the **rows below in this table** (gap clocks, current address). If gap mode is set to “E2925A compatibility”, information is as described in the **tables above** (signals, states, etc.). The gap mode is set by the “performance analyzer mode” property.
B_SIG_gap_clocks	gap_clocks	number	Number of clocks during which storage was suppressed.
B_SIG_crr_addrlo B_SIG_crr_addrhi	crr_addrlo crr_addrhi	10 vector	Current address (high and low 32 bits, the upper bits are available only with a 64-bit testcard) , that is the address of the data phase that follows the gap. This allows address information even if data phases only are sampled (address phases are suppressed by storage qualification).

b_sizetype

size (CLI Abbreviation)
B_SIZE_BYTE (1)
B_SIZE_WORD (2)
B_SIZE_DWORD (4)

b_systeminfotype

infoprop (CLI Abbreviation)	value	Description
B_SINFO_BUSWIDTH (width)	32, 64	PCI bus width: 32 or 64 bit
B_SINFO_BUSSPEED (speed)	33000, 66000	PCI bus speed in Hz
B_SINFO_CPCI_GEOADDR (cpciga)	1 … 32	Provides the geographical address of the card within a CompactPCI system. For information, refer to the CompactPCI Hot Swap Specification.

b_targetgenproptype

targetgenprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_TGEN_BACKCAPABLE (backcapable)	Sets the Fast Back-to-Back capability bit in the command register of the configuration space (bit 9).	
	default: 0	Disables Fast Back-to-Back.
	1	Enables Fast Back-to-Back.
B_TGEN_IOSPACE (iospace)	Sets the I/O decoder enable bit in the command register of the configuration space (bit 0). Normally the bit is set during the system configuration routine to enable the decoder after system initialization.	
	default: 0	Disables I/O decoders.
	1	Enables I/O decoders.
B_TGEN_MEMSPACE (memspace)	Sets the memory decoder enable bit in the command register of the configuration space (bit 1). Normally the bit is set during the system configuration routine to enable the decoder after system initialization.	
	default: 0	Disables memory decoders.
	1	Enables memory decoders.
B_TGEN_ROMENABLE (romenable)	Sets the Expansion ROM decoder enable bit in the expansion ROM register of the configuration space (bit 0). Normally the bit is set during the system configuration routine to enable the decoder after system initialization.	
	default: 0	Disables Expansion ROM.
	1	Enables Expansion ROM decoder.
B_TGEN_RUNMODE (runmode)	B_RUNMODE_ADDRRESTART (addrrestart, 0)	Sets the target to restart from the beginning of the attribute page with every address phase.
	default: B_RUNMODE_ SEQUENTIAL (sequential, 1)	Sets the target to loop the attribute page only at the end of the page, thus providing a kind of random attribute set with respect to the address phases.

b_tattrgrouptype

NOTE The target attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

Target Attribute Group	group (CLI Abbreviation)	Target Attributes in the Group (CLI Abbreviation)
Address Phase Attributes	B_TATTR_GRP_TA0 (tagrpa0)	B_T_ACK64 (ack64) B_T_APERR (aperr) B_T_DACPERR (dacperr) B_T_DOLOOP* (loop)
Data Phase Attributes	B_TATTR_GRP_TD0 (tagrpd0)	B_T_WAITS (w) B_T_DOLOOP* (loop)
	B_TATTR_GRP_TD1 (tagrpd1)	B_T_TERM (term) B_T_DPERR (dperr) B_T_DSERR (dserr) B_T_WRPAR (wp) B_T_WRPAR64 (wp64) B_T_DOLOOP* (loop)
	B_TATTR_GRP_TD2 (tagrpd2)	B_T_MARKER* (mark) B_T_DOLOOP* (loop)
Control Attributes	B_TATTR_GRP_TC (tagrpc)	B_T_REPEAT* (repeat) B_T_DOLOOP* (loop)

b_tattrproptype

The target attribute properties of type b_tattrproptype are divided into the following sections:

- *"Address Phase Attributes (Target)" on page 278*
- *"Data Phase Attributes (Target)" on page 279*
- *"Control Attributes (Target)" on page 281*

Address Phase Attributes (Target)

tattrprop (CLI Abbreviation)	value	Description
B_T_ACK64 (ack64)	**default:** 0	64-bit requests are not acknowledged.
	1	64-bit requests are acknowledged.
B_T_APERR (aperr)	Asserts a system error (SERR#) two clocks after the address phase (SERR# is used to signal address parity errors). This property also sets the SERR flag in the configuration space and in the status register of the testcard. System errors must be enabled in the configuration space for this property to have an effect. **Note:** If the subtractive decoder is enabled, SERR# is asserted, although the transaction can later be claimed by another device.	
	default: 0	No error is signaled.
	1	Error is signaled.
B_T_DACPERR (dacperr)	Asserts a system error (SERR#) two clocks after the second cycle of a dual address cycle (SERR# is used to signal address parity errors). This property also sets the SERR flag in the configuration space and in the status register of the testcard. System errors must be enabled in the configuration space for this property to have an effect. **Note:** Be cautious if the subtractive decoder is enabled. SERR# is asserted, although the transaction can later be claimed by another device.	
	default: 0	No error is signaled.
	1	Error is signaled.

Data Phase Attributes (Target)

NOTE The target attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

tattrprop (CLI Abbreviation)	value	Description
B_T_DPERR (dperr)	Asserts a parity error (PERR#) two clocks after the write data transfer, along with TRDY# or STOP#. This property also sets the PERR flag in the configuration space and in the status register of the testcard. Parity errors must be enabled in the configuration space for this property to have an effect. **Note:** This attribute is ignored both in read transfers and in write transfers when data comparison is enabled.	
	default: 0	No error is signaled.
	1	Error is signaled.
B_T_DSERR (dserr)	Asserts a system error (SERR#) two clocks after the data phase and sets the SERR flag in the configuration space and in the status register of the testcard. System errors must be enabled in the configuration space for this property to have an effect. **Note:** If the subtractive decoder is enabled, SERR# is asserted, although the transaction can later be claimed by another device. That device asserts DEVSEL# and TRDY# or STOP#.	
	default: 0	No error is signaled.
	1	Error is signaled.
B_T_MARKER* (marker)	0 … 15 **default:** 0	Number issued during address and data phase. It can be observed by pattern terms, for example, for synchronization. By convention, the value “0” is used to indicate “nothing special”.

tattrprop (CLI Abbreviation)	**value**	**Description**
B_T_TERM (term)	Specifies a termination applied after the number of clocks as specified by B_T_WAITS.	
	B_T_NOTERM (noterm)	Data is accepted. No termination.
	B_T_TERM_RETRY (retry)	Data is not accepted. A disconnect-C or retry—depending on the time within the burst—is signaled.
	B_T_DISCONNECT (discon)	Data is accepted. A disconnect-A or disconnect-B—depending on the time within the burst—is signaled.
	B_T_ABORT (abort)	A target abort is signaled.
B_T_WAITS	Number of waits. This is controlled by the TRDY# behavior per data phase.	
	B_T_WAITS_HANG (whang), –1	Simulates a hanging target (TRDY# is not asserted).
	0 … 30 **default:** 0	Number of waits inserted. In the first data phase, this value refers to the number of clocks between the end of the address phase and the assertion of TRDY#. If prefetching was successful, 0 waits can be achieved, otherwise at least 7 waits are inserted.
	31	For compatibility reasons only. 30 waits are generated.
B_T_WRPAR (wp)	A wrong parity is set one clock after a read transfer. This attribute is ignored in write transfers.	
	default: 0	Parity remains as it is.
	1	Parity is inverted.
B_T_WRPAR64 (wp64)	A wrong parity (PAR64) is set one clock after a read transfer. This attribute is ignored in write transfers.	
	default: 0	Parity remains as it is.
	1	Parity is inverted.

Control Attributes (Target)

NOTE The target attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

tattrprop (CLI Abbreviation)	value	Description
B_T_DOLOOP* (loop)	Loop bit. This bit indicates the last line in the attribute page. The next line executed after this one is again the first line of the same page. Used with *"BestTargetAttrGroupLineProg" on page 129*, the loop bit is set for the referring target attribute group. Used with *"BestTargetAttrLineProg" on page 131*, the *general loop bit* for the complete page is set (previously set group loop bits of this page are overwritten).	
	default: 0	Continue with the next attribute line.
	1	Continue with the first line of the same page.
B_T_REPEAT* (repeat)	One attribute line can be performed repeatedly.	
	0	This line is used for all transfers.
	1 … 2^{32} **default:** 1	Number of repetitions.

b_tcgenproptype

tcgenprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_TCGEN_SPEC (spec)	**default:** 1	PCI Specification determines the permissible values of setup and hold time (preparation register values are ignored).
	0	Preparation register values can be written to the testcard.

b_tcproptype

tcprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_TC_SETUP_TIME (setup)	With a 29 – 35 MHz PCI bus: 5000 … 9000 **default:** 7000 With a 35 – 67 MHz PCI bus: 1000 … 5000 **default:** 3000	Setup time t_{su} in ps (picoseconds). The value must be divisible by 250.
B_TC_HOLD_TIME (hold)	0 … 2000 **default:** 0	Absolute value of the hold time t_h. The value must be divisible by 250.
B_TC_HSIGN (hsign)	0, 1 **default:** 0	Sign of the hold time t_h. 1 = negative hold time.

b_tcstatustype

status (CLI Abbreviation)	value	Description
B_TC_TCSTAT (tcstat)	B_TC_VIOLATION (0x2)	Bit 1 indicates whether a timing violation has occurred.
	B_TC_ERROR (0x1)	Bit 0 indicates whether the timing check works properly. **Note:** The current timing check results are invalid if this error flag is set.

b_testproptype

testprop (CLI Abbreviation)	value (CLI Abbreviation)	Desfription
B_TST_BANDWIDTH (bw)	0 … 100 **default:** 50	Requested bus bandwidth in percent. The test tries to occupy the bus with this bandwidth. **Note:** Bandwidth values of more than 95 % may slow down the device under test so much that it appears to be hanging.
B_TST_BLKLENGTH (blklen)	1 … 64k (dword aligned) **default:** 8192	Number of bytes for a data block. The tests transfer data blockwise.
B_TST_COMPARE (comp)	0, 1 **default:** 0	Enable/disable data compare between written and read data.
B_TST_DATAPATTERN (dpat)	Sets a data pattern for the tests. The patterns are generated during the test run. The testcard's internal data memory is not used.	
	default: B_DATAPATTERN_RANDOM (dprando)	Random data pattern.
	B_DATAPATTERN_FIX (dpfix)	Data is always 00000000\h.
	B_DATAPATTERN_TOGGLE (dptoggle)	Data toggles between 00000000\h and FFFFFFFF\h.
B_TST_DESTINADDR (dest)	32 Bit, dword aligned **default:** 00000000\h	PCI address of source data. Used when moving blocks within PCI system.
B_TST_NOFBYTES (nob)	32 Bit **default:** 4	Number of transferred bytes.

testprop (CLI Abbreviation)	value (CLI Abbreviation)	Desfription
B_TST_PROTOCOL (prot)	Sets the protocol stress level.	
	default: B_PROTOCOL_LITE (lite)	Stresses the system as little as possible.
	B_PROTOCOL_MEDIUM (medium)	Generates medium protocol stress.
	B_PROTOCOL_HARD (hard)	Generates maximum protocol stress.
B_TST_STARTADDR (start)	32 Bit, dword aligned **default:** 00000000\h	PCI address of the memory range to be accessed by a test (except block move).
B_TST_SOURCEADDR (source)	32 Bit, dword aligned **default:** 00000000\h	PCI address of source data. Used when moving blocks within PCI system.

b_tracepattproptype

tracepattprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_PT_TRIGGER (trig)	“<pattern_string>”	Trigger pattern.
B_PT_SQ (sq)	“<pattern_string>”	Storage qualifier.

NOTE After initialization, the properties may hold any value because the properties are not preset to default values. This can affect your measurement. Therefore, set both properties with *“BestTracePattPropSet” on page 82*.

To build pattern terms, refer to *“Pattern Term Identifiers” on page 65*.

b_traceproptype

traceprop (CLI Abbreviation)	value (CLI abbreciation)	Description
B_TRC_HEARTBEATMODE (hbmode)	B_HBMODE_ON (on, 1)	Heartbeat trigger mode.
	default: B_HBMODE_OFF (off, 0)	Normal trigger mode.
B_TRC_HEARTBEATVALUE (hbvalue)	32 bits	Heartbeat trigger value in PCI clocks.
B_TRC_PERFANALYZER_MODE (amode)	**default:** B_E2925_COMPATIBLE (e2925, 0)	Only the gap flag indicates a gap. Programs written for the trace memory of the Agilent E2925A run if this is activated.
	B_PERFORMANCE (perf, 1)	Exhaustive gap information is stored (normal gap mode).
B_TRC_PATT0_MODE (pt0mode)	**default:** B_PATT0MODE_STANDARD (standard, 1)	Pattern term pt0 is used as a standard pattern term.
	B_PATT0MODE_DIFFERENTIAL (differential, 1)	Pattern term pt0 is used as a transitional (same as: differential) pattern term.
B_TRC_TRIG_HISTORY (trighist)	0 … ffff0\h in steps of 4 ; The default depends on the available trace memory space: it is set so that the trigger can be found in the center of the trace memory.	Trigger counter preload value defining how many lines are captured after a trigger event. Only multiples of 4 are allowed.

b_tracestatustype

tracestatus (CLI Abbreviation)	Description
B_TRC_STAT (stat)	Contents of the trace status register; see table below.
B_TRC_TRIGPOINT (trig)	Line number corresponding to the trigger event.
B_TRC_LINESCAPT (lines)	Number of lines captured.

Trace Status Register

The following table shows the meanings of the single bits of the trace status register:

Bit	Meaning	Description
0	1 = Trace memory stopped.	
1	1 = Trigger occurred.	
2	0 = B_E2925_COMPATIBLE 1 = B_PERFORMANCE	Current mode of the performance analyzer. See also B_TRC_PERFANALYZER_MODE of *"b_traceproptype" on page 285*.
3 … 31	Not used.	

b_trigioseqgenproptype

trigiogenprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_TRIGIOSEQGEN_OUT_0 (io0) … B_TRIGIOSEQGEN_OUT_11 (io11)	Sets the output mode of trigger line 0 … 11. (Input is always possible.)	
	B_TRIGIO_INPONLY (inponly)	Disables the output.
	B_TRIGIO_TOTEMPOLE (totempole)	Sets the output to totem-pole.
	B_TRIGIO_OPENDRAIN (opendrain)	Sets the output to open-drain.
B_TRIGIOSEQGEN_CTRC_PREL (cprel)	0 … $2^{32} - 1$ **default:** 0	Preload value for feedback counter C.

b_trigioseqtrancondproptype

trigioseqgenprop (CLI Abbreviation)	default value (CLI Abbreviation)	Description
B_TRIGIOSEQ_XCOND (x)	"1" (true)	Transition condition for the trigger I/O sequencer to move from one state to the next.
B_TRIGIOSEQ_OUT_0 (out0) … B_TRIGIOSEQ_OUT_11 (out11)	"0" (false)	Output condition on trigger I/O pins 0 …11.
B_TRGIOSEQ_CDEC (cdec)	"0" (false)	Condition to decrement feedback counter C.
B_TRIGIOSEQ_DDEC (ddec)	"0" (false)	Condition to decrement feedback counter D
B_TRIGIOSEQ_CINC (cinc)	"0" (false)	Condition to increment feedback counter C
B_TRIGIOSEQ_DINC (dinc)	"0" (false)	Condition to increment feedback counter D
B_TRIGIOSEQ_CLOAD (cload)	"0" (false)	Condition to preload feedback counter C.
B_TRIGIOSEQ_DLOAD (dload)	"0" (false)	Condition to preload feedback counter D

b_trigioseqtranproptype

trigioseqtranprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_TRIGIOSEQ_STATE (state)	0 … 63 **default:** 0	State identifier. The sequencer starts at state 0.
B_TRIGIOSEQ_NEXTSTATE (nextstate)	0 … 63 **default:** 0	Identifier for the state after the transition condition has been met.

b_trigseqgenproptype

trigseqgenprop (CLI Abbreviation)	value (CLI Abbreviation)	Description
B_TRIGSEQGEN_CTRC_PREL (cprel)	0 … $2^{32} - 1$ **default:** 0	Preload value for feedback counter C.

b_trigseqtrancondproptype

trigseqtrancondprop (CLI Abbreviation)	Default values (CLI Abbreviation)	Description
B_TRIGSEQ_XCOND (x)	"1" (true)	Transition condition for the trigger sequencer to move from one state to the next.
B_TRIGSEQ_TRIGCOND (trig)	"0" (false)	Trigger condition.
B_TRIGSEQ_SQCOND (sq)	"0" (false)	Condition to store the current trace data line in the trace memory (Storage Qualifier Condition).
B_TRGIOSEQ_CDEC (cdec)	"0" (false)	Condition to decrement feedback counter C.
B_TRIGIOSEQ_DDEC (ddec)	"0" (false)	Condition to decrement feedback counter D
B_TRIGIOSEQ_CINC (cinc)	"0" (false)	Condition to increment feedback counter C
B_TRIGIOSEQ_DINC (dinc)	"0" (false)	Condition to increment feedback counter D
B_TRIGIOSEQ_CLOAD (cload)	"0" (false)	Condition to preload feedback counter C.
B_TRIGIOSEQ_DLOAD (dload)	"0" (false)	Condition to preload feedback counter D

b_trigseqtranproptype

trigseqtranprop (CLI Abbreviation)	value	Description
B_TRIGSEQ_STATE (state)	0 … 63 **default:** 0	State identifier. The sequencer starts at state 0.
B_TRIGSEQ_NEXTSTATE (nextstate)	0 … 63 **default:** 0	Identifier for the state after the transition condition has been met.

b_versionproptype

versionprop (CLI Abbreviation)	Description
B_VER_BOARD (board)	Returns the testcard ID (for example, B3835-0069).
B_VER_CAPI (capi)	Returns the C-API version.
B_VER_CORE (core)	Returns the date code of the onboard core BIOS.
B_VER_FIRMWARE (firmware)	Returns the version code of the onboard firmware.
B_VER_FIRMWARE_DATE (firmwaredate)	Returns the version date of the onboard firmware.
B_VER_PRODUCT (product)	Returns the testcard product number (for example, E2940A).
B_VER_SERIAL (serial)	Returns the serial number of the testcard hardware.
B_VER_TEAM (team)	Returns the members of the developer team.
B_VER_XILDATE (xildate)	Returns the date code of the Xilinx FPGA chain architecture file.

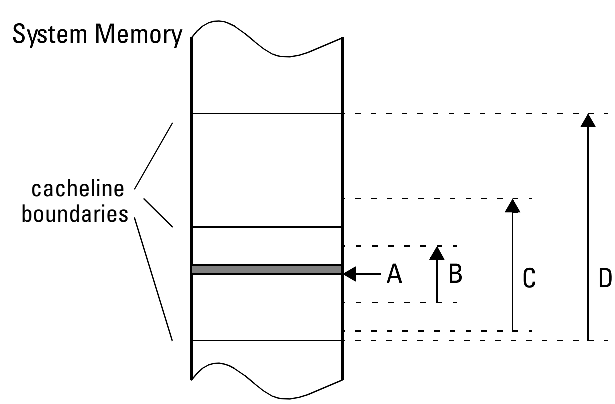
bppr_algorithmtype

Values (CLI Abbreviation)	Description
default: BPPR_ALG_PERM (perm)	All values are picked one after the other as listed in the value list of the referring function.
BPPR_ALG_RAND (rand)	Values are picked at random from the value list. Using this algorithm cannot guarantee that **all** values are picked.
BPPR_ALG_RECOMM (recomm)	Only for the block variation parameter B_BLK_CMDS. The PCI bus commands from the value list will be picked one after the other, but the algorithm honors PCI Specification recommendations (see description below).
BPPR_ALG_BEST (best)	Only for the block variation parameter B_BLK_CMDS. Only best suitable PCI bus commands from the value list will be chosen for each permutation (see description below).

bppr_algorithmtype: B_BLK_CMDS Details

For the block variation parameters additional algorithms are available. Here the selection of parameters can be constrained to recommended or best suitable PCI bus commands.

When using PCI bus commands to perform memory write or read transfers, the following relationships between block start and end address and cacheline boundaries can occur and influence the commands usage:



Scenario	Description
A	Spans one dword only.
B	Spans multiple dwords, but range does not cross any cacheline boundary.
C	Spans multiple dwords, and crosses at least one cacheline boundary.
D	Spans multiple dwords. Start and end address are on a cacheline boundary.

Possible Commands

The following table shows, which commands the PCI PPR software can choose, depending on the specified algorithm and a specific scenario.

		Algorithm		
Direction	Scenario	PERM, RAND	RECOMM	BEST
READ	A	MR, MRL, MRM	MR	MR
	B	MR, MRL, MRM	MR, MRL	MRL
	C, D	MR, MRL, MRM	MR, MRL, MRM	MRM
WRITE	all except D	MW	MW	MW
	D	MW, MWI	MW, MWI	MWI

In this table the following abbreviations are used for the PCI memory commands—numbers in brackets are their decimal equivalents:

- MR – Read (6)
- MRL – Read Line (14)
- MRM – Read Multiple Lines (12)
- MW – Write (7)
- MWI – Write & Invalidate (15)

bppr_blkpermproptype

The following table lists the available block permutation properties and the appropriate value ranges. Values in these ranges can be used in the corresponding value lists.

prop (CLI Abbreviation)	**value/ranges**	**Description**
BPPR_BLK_DIR (dir)	B_DIR_WRITE	Transfer from the testcard’s internal to system memory.
	default: B_DIR_READ	Transfer from system to the testcard’s internal memory.
BPPR_BLK_BUSADDR (busaddr)	32-bit integer **default:** 0	Start address on the bus for the compound block transfer.
BPPR_BLK_BUSADDR_HI (busaddrhi)	32-bit integer **default:** 0	Upper 32 bits of a 64-bit bus address (requires BPPR_BLK_BUSDAC to be enabled).
BPPR_BLK_BUSDAC (busdac)	Determines whether 32-bit or 64-bit addresses are transferred during an address phase.	
	default: 0	Single address cycle (SAC) with 32-bit address.
	1	Dual address cycle (DAC) with 64-bit address. Uses BPPR_BLK_BUSADDR_HI.
BPPR_BLK_INTADDR (intaddr)	0 … 0xFFFFC **default:** 0	Start address in the testcard’s internal data memory for the compound block transfer. Refer to *“b_blkproptype” on page 228*.
BPPR_BLK_NOFDWORDS (nod)	31-bit value **default:** 1	Compound block size. Specifies the total number of dwords to be transferred.
BPPR_BLK_ATTRPAGE (apage)	0 … 255 **default:** 1	Specifies the master attribute page used for the compound block transfer (refer to *“b_blkproptype” on page 228*). With each data phase in the compound block the lines of this attribute page are sequentially executed. To determine when the line pointer should be reset to the start of the attribute page, the property B_MGEN_ATTRMODE of *“b_mastergenproptype” on page 248* can be used.
BPPR_BLK_COMPFLAG (compflag)	0, 1 **default:** 0	Refer to *“b_blkproptype” on page 228*.
BPPR_BLK_COMPOFFS (compoffs)	0 … 0xFFFFC **default:** 0	Refer to *“b_blkproptype” on page 228*.
BPPR_BLK_PAGENUM (pagenum)	0 … 15 **default:** 0	Block page number.
BPPR_BLK_PAGESIZEMAX (pagesizemax)	1 … 256 **default:** 1	Maximum page size, measured in blocks.

<table>
<tr><th>prop
(CLI Abbreviation)</th><th>value/ranges</th><th>Description</th></tr>
<tr><td>BPPR_BLK_FIRSTPERM
(firstperm)</td><td>1 … 10^9
default: 1</td><td>Master block first permutation number.

Starts the permutation algorithm at the specific value. If not all desired permutations have been exercised previously, you can continue the permutation by setting this number to where the last permutation ended.
The number of the last permutation can be queried with BPPR_BLK_LASTPERM; see “bppr_blkresultparamtype” on page 296.</td></tr>
<tr><td rowspan="3">BPPR_BLK_CACHELINE
(cacheline)</td><td colspan="2">Informs the block permutator about the system’s cacheline size, which is required to know which PCI bus commands are legal.</td></tr>
<tr><td>default:
BPPR_BLK_CACHELINE_NO</td><td>The system’s cacheline size is either unknown or no cache exists.</td></tr>
<tr><td>A power of 2 between 2 and 2048.</td><td>Cacheline size, measured in dwords.</td></tr>
<tr><td rowspan="3">BPPR_BLK_FILLGAPS
(fillgaps)</td><td colspan="2">Determines whether or not remaining gaps are filled after fitting block permutations into the compound block size.

Filling the gaps guarantees that the whole block is transferred. On the other hand, filling gaps may also require that address alignments or byte enable values are used that are (perhaps intentionally) not specified in “BestPprBlockVariationSet” on page 199.</td></tr>
<tr><td>default: 1</td><td>All bytes are transferred. Unlisted variation values may be used.</td></tr>
<tr><td>0</td><td>No unlisted variation values are used. Not all bytes may be transferred.</td></tr>
</table>

bppr_blkresultparamtype

param (CLI Abbreviation)	Description
BPPR_BLK_PAGESIZEACT (pagesizeact)	Number of blocks that have actually been used to implement the compound block (that is the master block page size).
BPPR_BLK_LASTPERM (lastperm)	Number of the last permutation of block variation parameters that did fit into the compound block. This number can be used to determine which permutations will be covered after complete execution of the compound block.
BPPR_BLK_TIME (time)	Estimated testing time in µs required to execute the compound block once. This does not include the initial programming time. To determine the testing time, the generic properties BPPR_GEN_BUSSPEED and BPPR_GEN_XFERCLKS of *"bppr_genproptype" on page 300* are evaluated.

bppr_blkvarparamtype

The following table lists the available block variation parameters and the corresponding value ranges. Values within these ranges can be used in the corresponding value lists. A syntax description for the entries of a value list can be found in *"Value Lists" on page 298*.

param (CLI Abbreviation)	**value ranges for entries in value_lists**	**Description**
BPPR_BLK_ALIGN (align)	**Granularity:** Power of 2 between cacheline size and 8192. **Offset:** Multiple of 4 between 0 and 8188. **default:** (%32=0)	Restricts the start of a block to selected offsets using a given granularity. For example, "(%16=4)" means 4 bytes after a 16 byte boundary (address modulo 16 must be 4). The generation algorithm will perform a check against the specified cacheline size.
BPPR_BLK_BYTEN (byten)	0 … 15 **default:** 0	Numeric values for the C/BE lines in the address phase. Be aware that gaps may occur if BPPR_BLK_FILLGAPS of *"bppr_blkpermproptype" on page 294* is set to "0".
BPPR_BLK_CMDS (cmds)	int_ack special io_read io_write reserved_4 reserved_5 **default:** mem_read **default:** mem_write reserved_8 reserved_9 config_read config_write mem_readmultiple mem_dac mem_readline mem_writeinvalidate	List of PCI bus commands to be used for variations. Only suitable values according to specified transfer direction (BPPR_BLK_DIR) will be used. Therefore, the default value also depends on "BPPR_BLK_DIR". Refer to *"bppr_blkpermproptype" on page 294.* Make sure, that you use extended read commands in prefetchable area only. For more information on the usage of the PCI bus commands, refer to *"bppr_algorithmtype: B_BLK_CMDS Details" on page 292.*
BPPR_BLK_SIZE (size)	Multiple of 4 between 4 and 128k. **default:** 4	Numeric value for a block size, measured in bytes.

Value Lists

Content The parameter value_list holds the values to be used for permutation or randomization. They must always be suited to the corresponding variation parameter.

Values may be held in the list repeatedly:

- to determine how many times this value will appear in the generated tuples.

  For example, if a value is held twice in the list, it will appear twice as much as values that appear once in the list.

- to generate an individual sequence of tuples, in which this value should appear repeatedly.

  For this purpose the algorithm BPPR_ALG_PERM is used. See *"bppr_algorithmtype" on page 292*.

Syntax value_list is a quoted string with a comma-separated list of values (non case-sensitive). The format of a valid value entry depends on the variation parameter.

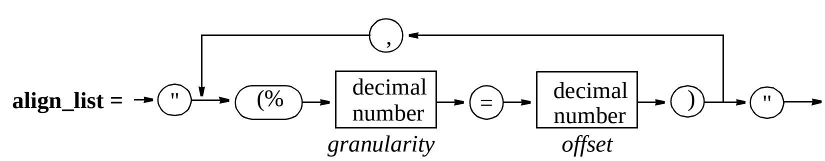
Example:

- `"io_read, mem_read, mem_readmultiple, mem_readline"`

There are two different kinds of values. Alignment values and numeric values.

Alignment values

Alignment values must be given in brackets. They start with a percent sign followed by the granularity, an equality sign, and the offset. This syntax is shown in the following diagram:



Example:

- `"(%4=0)"`

  Means a granularity of 4 and an offset of 0 bytes.

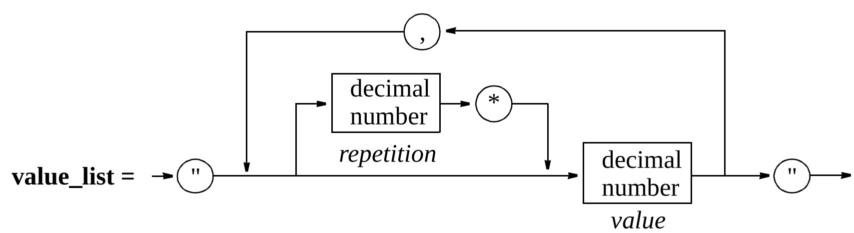
NOTE For more information on granularity, offset and value, see *"bppr_blkvarparamtype" on page 297*.

Numeric values

Numeric values can be commands, byte enables and block sizes. In this case the value list is a quoted string consisting of a comma-separated list of **symbols** or **numbers**, for example, `"B_RELREQ_OFF, 2, 4, 6, 8"`.

Use given symbols instead of numbers whenever possible. If using numbers, they must be specified in decimal number format.

The syntax for value lists is shown in the following diagram—a repetition factor can be given to repeat a value in the list:



Example:

- `"99*0,1"`

  Generates a list of 100 entries with 99 zeros and one 1. In one percent of all cases the attribute value is set to 1.

NOTE *Repetition* values increase the actual length of the list. This length must be less than 100 entries.

bppr_genproptype

prop (CLI Abbreviation)	value	Description
BPPR_GEN_BUSSPEED (busspeed)	1 … 66 **default:** 33	PCI bus speed in MHz. Used to calculate times from clock cycles.
BPPR_GEN_BUSWIDTH (buswidth)	32, 64 **default:** 32	PCI bus width of the used system in bits.
BPPR_GEN_SEED (seed)	32-bit integer **default:** 0	Start value for the generation of pseudo random number sequences used by the permutations algorithm. Use this to make “random” numbers reproducible.
BPPR_GEN_XFERCLKS (xferclks)	1 … 1000 **default:** 10	Expected number of clocks per data transfer. Used to estimate testing times.

bppr_mattrpermproptype

prop (CLI Abbreviation)	value	Description
BPPR_MA_FIRSTPERM (firstperm)	1 … 10^9 **default:** 1	Master attribute first permutation number. Starts the permutation algorithm at the specific point. If not all desired permutations have been exercised previously, you can continue the permutation by setting this number to where the last permutation ended. The last permutation can be queried from property BPPR_MA_LASTPERM. Refer to *"bppr_mattrresultparamtype" on page 302*.
BPPR_MA_PAGENUM (pagenum)	1 … 255 **default:** 1	Master attribute page number to be used for the permutation of attributes.
BPPR_MA_PAGESIZEMA X (pagesizemax)	0 … 255 **default:** 1	Maximum master attribute page size, measured in attribute lines. All desired master attribute permutations must fit into this page.
BPPR_MA_TUPLES (tuples)	0 … 9 **default:** 3	Maximum number of groups that are permutated against each other for calculation of coverage.

bppr_mattrresultparamtype

param (CLI Abbreviation)	value	Description
BPPR_MA_DATA (data)	0 … 2^{32}	Maximum amount of data in bytes that must be transferred to achieve complete coverage of master attribute variations. Depends on the buswidth. See BPPR_GEN_BUSWIDTH of type *"bppr_genproptype" on page 300*.
BPPR_MA_LASTPERM (lastperm)	0 … 2^{32}	Number of the last permutation of master attributes in the allocated attribute page. This number can be used to determine which permutations will be covered after complete execution of the attribute page.
BPPR_MA_PAGESIZEACT (pagesizeact)	0 … 255	Actually used attribute page size. This size is always below the value specified by BPPR_MA_PAGESIZEMAX.
BPPR_MA_TIME (time)	0 … 2^{32}	Estimated testing time in µs required to go through the complete attribute page. This does not include the initial programming time. To determine the testing time, the general properties BPPR_GEN_BUSSPEED and BPPR_GEN_XFERCLKS of *"bppr_genproptype" on page 300* are evaluated.
BPPR_MA_TUPLES_DATA (tuplesdata)	0 … 2^{32}	Maximum amount of data in bytes that must be transferred to achieve complete coverage of all required group tuples. See BPPR_MA_TUPLES of type *"bppr_mattrpermproptype" on page 301*.
BPPR_MA_TUPLES_TIME (tuplestime)	0 … 2^{32}	Estimated time in µs required to transfer enough data to achieve complete coverage of all required group tuples. See BPPR_MA_TUPLES of type *"bppr_mattrpermproptype" on page 301*.
BPPR_MA_RUNS (runs)	0 … 2^{32}	Number of block page runs needed for complete coverage if block variations are set to be permutated.
BPPR_MA_TUPLES_RUNS (tuplesruns)	0 … 2^{32}	Number of block page runs needed for group tuple coverage if block variations are set to be permutated.

bppr_reportproptype

prop (CLI Abbreviation)	value	Description
BPPR_REP_BLOCK (block)	0, 1 **default:** 1	Report of block permutation.
BPPR_REP_MA (ma)	0, 1 **default:** 1	Report of master attribute permutation.
BPPR_REP_REPORT (report)	0, 1 **default:** 1	Report of report properties.
BPPR_REP_TA (ta)	0, 1 **default:** 1	Report of target attribute permutation.
BPPR_REP_ORDER_TUPLE (order)	1 … 9 **default:** 3	Specifies how many elements of a tuple are used for coverage report (maximum tuple order). “1” means that the coverage for individual attributes is reported, “2” means that the coverage for pairs of attributes is reported, and so on. Valid for reporting of block, master and target permutations.
BPPR_REP_CAPI (capi)	0, 1 **default:** 1	Includes C-API abbreviations into the report.
BPPR_REP_BLOCKCONTENT (blockcont)	0 … 256 **default:** 30	Specifies the maximum number of lines of generated permutations to be reported.
BPPR_REP_MACONTENT (macont)	0 … 2^{32} **default:** 30	Specifies the maximum number of master attribute page lines to be reported, starting with first page line.
BPPR_REP_TACONTENT (tacont)	0 … 2^{32} **default:** 30	Specifies the maximum number of target attribute page lines to be reported, starting with the first page line.

bppr_tattrpermproptype

prop (CLI Abbreviation)	**value**	**Description**
BPPR_TA_FIRSTPERM (firstperm)	1 … 10^9 **default:** 1	Target attribute first permutation. Starts the permutation algorithm at the specific point. If not all desired permutations have been exercised previously, you can continue the permutation by setting this number to where the last permutation ended. The last iteration can be queried from property BPPR_TA_LASTPERM. Refer to *"bppr_tattrresultparamtype" on page 305*.
BPPR_TA_PAGENUM (pagenum)	1 … 255 **default:** 1	Target attribute page number to be used for permutation of attributes.
BPPR_TA_PAGESIZEMAX (pagesizemax)	0 … 255 **default:** 1	Maximum target attribute page size, measured in attribute lines. All desired target attribute permutations must fit into this page.
BPPR_TA_TUPLES (tuples)	0 … 5 **default:** 3	Maximum number of groups that are permutated against each other for calculation of coverage.

bppr_tattrresultparamtype

param (CLI Abbreviation)	value	Description
BPPR_TA_DATA (data)	0 … 2^{32}	Maximum amount of data in bytes that must be transferred to achieve complete coverage of target attribute variations. Depends on the buswidth. See BPPR_GEN_BUSWIDTH of type *"bppr_genproptype" on page 300*.
BPPR_TA_LASTPERM (lastperm)	0 … 2^{32}	Number of the last permutation of target attributes in the allocated attribute page. This number can be used to determine which permutations will be covered after complete execution of the attribute page.
BPPR_TA_PAGESIZEACT (pagesizeact)	0 … 255	Actually used attribute page size. This size is always below the value specified by BPPR_TA_PAGESIZEMAX.
BPPR_TA_TIME (time)	0 … 2^{32}	Estimated testing time in μs required to go through the complete attribute page. This does not include the initial programming time. To determine the testing time, the general properties BPPR_GEN_BUSSPEED and BPPR_GEN_XFERCLKS of *"bppr_genproptype" on page 300* are evaluated.
BPPR_TA_TUPLES_DATA (tuplesdata)	0 … 2^{32}	Maximum amount of data in bytes that must be transferred to achieve complete coverage of all required group tuples. See BPPR_TA_TUPLES of type *"bppr_tattrpermproptype" on page 304*.
BPPR_TA_TUPLES_TIME (tuplestime)	0 … 2^{32}	Estimated testing time in μs required to transfer enough data to achieve complete coverage of all required group tuples. See BPPR_TA_TUPLES of type *"bppr_tattrpermproptype" on page 304*.

Index

G

H

I

L

M

N

O

P

R

Publication Number: 5988-5055EN

Agilent Technologies