Agilent E2922A/B C-API/PPR

Programming Reference

Agilent Technologies

Important Notice

All information in this document is valid for both Agilent E2922A and Agilent E2922B testcards.

Copyright

Author: Anja Schauer, t3 medien GmbH

Notice

The material contained in this document is subject to change without notice. Agilent Technologies makes no warranty of any kind with regard to this material, 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 for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Contents

Standard Analysis Functions 41

Exerciser Functions 49

Introduction

The C-API/PPR Reference describes all C functions, types and definitions of the application programming interface of the Agilent PCI-X testcard. It also provides the commands and abbreviations that are used in the command line interface (CLI) of the graphical user interface (GUI).

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

This introduction contains the following sections:

- *"Differences between the PCI and the PCI-X C-API/PPR" on page 12* highlights the programming differences between PCI and PCI-X testcards. This is useful if you are already familiar with programming Agilent PCI testcards.

- *"Programming Interfaces" on page 13* introduces two ways of programming the PCI-X testcard. These are C-API and CLI.

- *"Naming Conventions" on page 14* introduces a consistent approach taking to naming constants, types, and functions. This is needed when documenting C functions.

- *"Structure of the PCI-X C-API/PPR Reference" on page 15* introduces the structure of this reference and the structures of individual sections.

Differences between the PCI and the PCI-X C-API/PPR

This section reviews the major differences between PCI and PCI-X testcard programming.

Getting Status Information

Status information can be obtained by using the single function call `BestXStatusRead`. This function allows you to read the board status and the exerciser status.

Changes in Terminology for the Exerciser

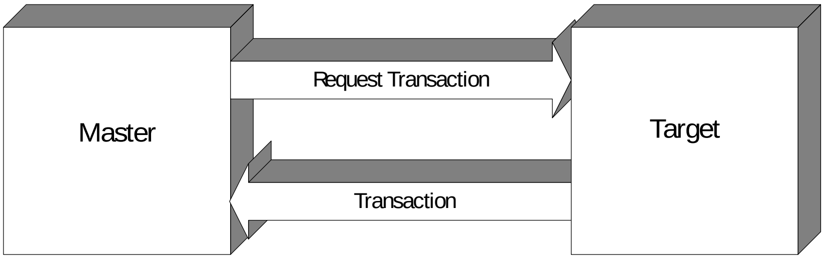
The PCI-X Specification introduces the following changes in terminology:

- "Attributes" has been renamed to "behavior" to avoid confusion with the PCI-X attribute phase.
- "Master" has been renamed to "Initiator".

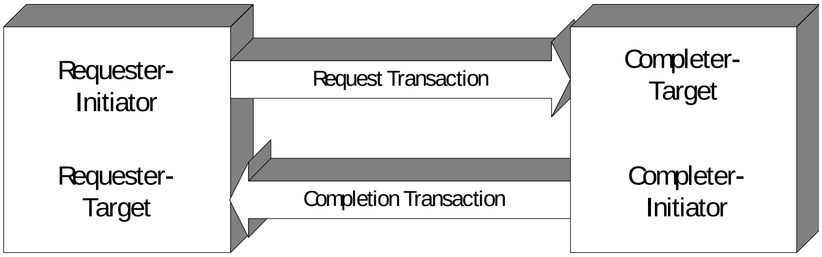
  The exerciser now consists of initiator and target.

  Because PCI-X allows you to perform split transactions, initiator and target have been subdivided. The changes are shown in the following figure:

  – PCI:



  – PCI-X:



Programming an Exerciser Memory

A major difference between PCI testcard programming and PCI-X testcard programming is in reading and writing to the types of exerciser memory:

- With the **PCI** C-API, you read and write to the memory by using a preparation register, which can hold all properties for one memory line.
- With the **PCI-X** C-API, you read and write to the memory directly. The properties for one line can be read and written by using a single function call.

Programming the Protocol Permutator and Randomizer

There are several changes compared with PPR programming for PCI testcards:

- A PPR session can be opened per testcard (per handle).
- Block permutation can also be performed for I/O accesses.
- All `BestPprxxxGenerate` functions are merged into `BestXPprProg`, which writes all current settings to the card.
- Only parts that are programmed to the testcard will appear in the report.
- Gap information can be retrieved and used for user-defined testing.

Programming Interfaces

You have two possibilities to program the Agilent PCI-X testcard:

C-API The application programming interface allows you control of the testcard by means of C function calls.

CLI The Command Line Interface 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.

Naming Conventions

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

Naming of Constants

Constants are written in upper case letters. In general, the constant name begins with `BX_`.

Example: `BX_RESLOCK_EXERCISER`.

Constants that are used exclusively in Protocol Permutator and Randomizer functions begin with `BXPPR_`.

Example: `BXPPR_GEN_BUSSPEED`

Naming of Types

Type names are written in lower case letters. In general, the type name begins with `bx_` and ends with `type`.

Example: `bx_resourcetype`

Types that are used exclusively by Protocol Permutator and Randomizer functions begin with `bxppr_`.

Example: `bxppr_gentype`

Naming of Functions

Function names are written in lower and upper case letters. In general, the function name begins with `BestX`. The action is indicated by the last words in the call.

Example: `BestXResourceLock`

Protocol Permutator and Randomizer functions begin with `BestXPpr`.

Example: `BestXPprGenSet`

Structure of the PCI-X C-API/PPR Reference

The PCI-X C-API/PPR Reference is divided into the following sections:

Sections	Contents
Generic Functions	These functions are used to control the connection and initialization as well as setup and status capabilities.
Standard Analysis Functions	Sets up the protocol observer.
Exerciser Functions	These functions are used to control the requester-initiator, the completer-initiator, the requester-target and the completer-target. Decoder and data memory access functions are also listed.
Protocol Permutator and Randomizer Functions	These functions set up and control parameter permutations for the requester-initiator, the completer initiator, the requester-target and the completer-target.
Error Handling	These functions support error tracing.
Type Definitions	Here you will find all type definitions that are used in the functions above.

Functions All sections except the type definition section are subdivided according to theme. Every group starts with an overview of the functions in logical order, followed by the full description of the functions in alphabetical order.

Type Definitions The type definitions section is not grouped according to subjects. All available type definitions are listed in alphabetical order.

Generic Functions

The PCI-X generic functions are divided as follows:

Generic Functions	Action
Initialization and Connection Functions	Connect and initialize the testcards.
Administration Functions	Inform about the system under test and lock and unlock resources.
Card Status Functions	Read and clear status registers.
Generic Control and Setup Functions	Write and read testcard and power-up properties.
Power Management Event Functions	Control power management events.
Interrupt Generation Function	Generates a PCI-X interrupt.

Initialization and Connection Functions

The following functions are used for connection and initialization:

Function	Action
"BestXDevIdentifierGet" on page 19	Returns the identifier of a PCI-X device.
"BestXOpen" on page 21	Opens a connection to the testcard.
"BestXClose" on page 18	Closes the connection to the testcard.
"BestXPing" on page 22	Checks a connection to the testcard.
"BestXRS232BaudRateSet" on page 22	Sets the baud rate if the serial interface is used.

BestXClose

Call

```
bx_errtype BestXClose ( bx_handletype handle );
```

Description Closes the connection to the testcard.

CLI Equivalent No CLI equivalent.

CLI Abbreviation No CLI abbreviation.

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also *"BestXOpen" on page 21*

BestXDevIdentifierGet

Call

```
bx_errtype BestXDevIdentifierGet(
      bx_int32     vendor_id,
      bx_int32     device_id,
      bx_int32     number,
      bx_int32     *devid );
```

Description Used if the PCI-X port is applied to 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-X bus. The returned device identifier can be used as port number for `BestXOpen`.

NOTE This function can only be used in systems with standard BIOS. For other systems, you must build the device identifier. See *"Device Identifier Format" on page 20*.

CLI Equivalent

```
BestXDevIdentifierGet vendor_id=<vendor_id> device_id=<device_id>
number=<number>
```

CLI Abbreviation

```
devidentifierget vendor=<vendor_id> dev=<device_id> n=<number>
```

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters

vendor_id Vendor ID of the testcard (default = 15BC\h).

device_id Device ID (vendor dependent) for the testcard (for example, 2929\h).

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

Output Parameters

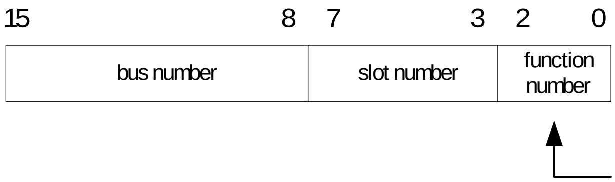
devid Device identifier within the system. This is the device number used to access the testcard's configuration space (for example, in *"BestXOpen" on page 21*).

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 `BestXDevIdentifierGet` to the `BestXOpen` call.

The function can only be used in systems with standard BIOS. For other systems, you must build the device identifier according to the following format rules:



Used to access functions of multi-function PCI-X devices. For single-function PCI-X devices, the function number is 0.

BestXOpen

Call

```
bx_errtype BestXOpen(
      bx_handletype  *handle,
      bx_porttype    port,
      bx_int32       portnum );
```

Description

Opens and checks the connection to the PCI-X 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 PCI-X connection).

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

CLI Equivalent No CLI equivalent.

CLI Abbreviation No CLI abbreviation.

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **port** Type of control port; see *“bx_porttype” on page 181*.

portnum Control port; see *“bx_porttype” on page 181*.

Output Parameters **handle** Session identification (comparable to a file handle).

See also *“BestXClose” on page 18*
“BestXDevIdentifierGet” on page 19

BestXPing

Call

```
bx_errtype BestXPing( bx_handletype handle );
```

Description Checks the connection to the testcard. If the connection is active and valid, the testcard responds with OK and both testcard LEDs flash simultaneously.

CLI Equivalent `BestXPing`

CLI Abbreviation `ping`

Return Value Error code; see *“bx_errtype” on page 176*.

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

See also –

BestXRS232BaudRateSet

Call

```
bx_errtype BestXRS232BaudRateSet(
      bx_handletype   handle,
      bx_int32        baudrate );
```

Description Defines the baud rate.

CLI Equivalent `BestXRS232BaudRateSet baudrate=<baudrate>`

CLI Abbreviation `rs232baudset baud=<baudrate>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

baudrate Baud rate value; see *“Baud Rate Values” on page 23*.

See also –

Baud Rate Values

Value	CLI Abbreviation	Baud Rate
BX_BD_9600	9600	9600 baud
BX_BD_19200	19200	19200 baud
BX_BD_38400	38400	38400 baud
BX_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	Action
"BestXVersionRead" on page 29	Checks the version of a component of the testcard.
"BestXCapabilityRead" on page 25	Reads the OR-combined capability code.
"BestXResourceLock" on page 27	Locks a resource.
"BestXResourceIsLocked" on page 26	Checks whether a resource is locked.
"BestXResourceUnlock" on page 28	Unlocks a resource.
"BestXAllResourceUnlock" on page 24	Unlocks all resources.

BestXAllResourceUnlock

Call

```
bx_errtype BestXAllResourceUnlock( bx_handletype handle );
```

Description Unlocks all locked resources (resets the internal counter incremented by each `BestXResourceLock` call). This function can be used to re-establish communication if the program hangs. If, for example, one controlling port hangs, you can regain control of the testcard without toggling the power.

CLI Equivalent `BestXAllResourceUnlock`

CLI Abbreviation `allresunlock`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also *"BestXResourceUnlock" on page 28*
"BestXResourceIsLocked" on page 26
"BestXResourceLock" on page 27

BestXCapabilityRead

Call

```
bx_errtype BestXCapabilityRead(
      bx_handletype   handle,
      bx_int32        *capa_code );
```

Description Reads the OR-combined capability codes.

CLI Equivalent `BestXCapabilityRead`

CLI Abbreviation `caparead`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

Output Parameters **capa_code** Value of the OR-combined capability codes as listed in the following table.

capa_code	Meaning
BX_CAPABILITY_EXERCISER	The testcard has the PCI-X exerciser capability.
BX_CAPABILITY_OBSERVER	The testcard has the observer with protocol rules enabled.
BX_CAPABILITY_CAPI	The capability allows you to program the testcard without using the graphical user interface.
BX_CAPABILITY_PERFORMANCE	The testcard supports performance counters.

See also –

BestXResourceIsLocked

Call

```
bx_errtype BestXResourceIsLocked(
      bx_handletype         handle,
      bx_bx_resourcetype    resource,
      bx_int32              *lock_count,
      bx_porttype           *lock_port );
```

Description Checks whether a resource has been locked during the current session.

CLI Equivalent `BestXResourceIsLocked resource=<resource>`

CLI Abbreviation `resislocked res=<resource>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

resource Resource to be checked; see *“bx_resourcetype” on page 181*.

Output Parameters

lock_count Number of resource locks. If the resource is unlocked, the value is 0.

lock_port Port that currently locks the resource; see *“bx_porttype” on page 181*. If `BX_PORT_CURRENT` is returned, the resource is locked by the current session.

See also

“BestXResourceUnlock” on page 28
“BestXAllResourceUnlock” on page 24
“BestXResourceLock” on page 27
“BestXPing” on page 22

BestXResourceLock

Call

```
bx_errtype BestXResourceLock(
     bx_handletype     handle,
     bx_resourcetype   resource );
```

Description

Locks a resource.

This function fails if the resource is already locked by another port. The function does not fail if the resource has already been locked by this port. Each time one resource is locked, an internal counter is incremented. The counter can be obtained using *“BestXResourceIsLocked” on page 26*.

CLI Equivalent

```
BestXResourceLock resource=<resource>
```

CLI Abbreviation

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

Return Value

Error code; see *“bx_errtype” on page 176*.

handle Session identification.

resource Resource to be locked; see *“bx_resourcetype” on page 181*.

See also

“BestXResourceUnlock” on page 28
“BestXAllResourceUnlock” on page 24

BestXResourceUnlock

Call

```
bx_errtype BestXResourceUnlock(
      bx_handletype    handle,
      bx_resourcetype  resource );
```

Description

Removes one resource lock.

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

CLI Equivalent

```
BestXResourceUnlock resource=<resource>
```

CLI Abbreviation

```
resunlock res=<resource>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

resource Resource to be unlocked; see *“bx_resourcetype” on page 181*.

See also

“BestXResourceIsLocked” on page 26
“BestXResourceLock” on page 27
“BestXAllResourceUnlock” on page 24

BestXVersionRead

Call

```
bx_errtype BestXVersionRead (
      bx_handletype    handle,
      bx_versiontype   prop
      bx_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 `BestXVersionRead prop=<prop>`

CLI Abbreviation `versionread prop=<prop>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

prop Version property to be read; see *"bx_versiontype" on page 200*.

Output Parameters

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

See also *"bx_versiontype" on page 200*

Card Status Functions

The following functions are used to access status register contents:

Function	Action
"BestXStatusRead" on page 31	Reads the content of a specific status register.
"BestXStatusClear" on page 30	Clears the contents of a status register.

BestXStatusClear

Call

```
bx_errtype BestXStatusClear(
        bx_handletype     handle,
        bx_statustype     prop);
```

Description Clears the content of a status register.

CLI Equivalent `BestXStatusClear prop=<prop>`

CLI Abbreviation `statusclear prop=<prop>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

prop Status register property. Only some of the status register properties can be used to clear a status. See *"bx_statustype" on page 196*.

See also *"BestXStatusRead" on page 31*

BestXStatusRead

Call

```
bx_errtype BestXStatusRead(
      bx_handletype     handle,
      bx_statustype     prop,
      bx_int32          *val );
```

Description

Reads the actual content of the selected status register. You can select between:

- testcard status
- exerciser status

CLI Equivalent

```
BestXStatusRead prop=<prop>
```

CLI Abbreviation

```
statusread prop=<prop>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

prop Status register property; see *"bx_statustype" on page 196*.

Output Parameters

val Status register content; see *"bx_statustype" on page 196*.

See also

"BestXStatusClear" on page 30

Generic Control and Setup Functions

The following functions are used to write and read testcard and power-up properties:

Function	**Action**
"BestXPUProg" on page 37	Stores the current setting of the configuration space into the non-volatile memory.
"BestXAllDefaultSet" on page 33	Resets all properties on the controlling host to default values.
"BestXBoardDefaultSet" on page 33	Sets the generic testcard properties to default values.
"BestXBoardSet" on page 37	Sets a generic testcard property.
"BestXBoardGet" on page 34	Gets a generic testcard property.
"BestXBoardReset" on page 36	Resets the testcard.
"BestXBoardRead" on page 36	Reads testcard properties from the testcard.
"BestXBoardProg" on page 35	Writes testcard properties to the testcard.
"BestXPMEWrite" on page 39	Issues a power management event.
"BestXPMERead" on page 38	Determines if a power management event has occurred.

BestXAllDefaultSet

Call `bx_errtype BestXAllDefaultSet( bx_handletype handle );`

Description Sets all properties on the host to default values.

CLI Equivalent `BestXAllDefaultSet`

CLI Abbreviation `alldefset`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

See also *“BestXPUProg” on page 37*

BestXBoardDefaultSet

Call `bx_errtype BestXBoardDefaultSet( bx_handletype handle );`

Description Sets the generic board properties on the host to default values. For board properties, see *“bx_boardtype” on page 160*.

To write these settings to the testcard, use the `BestXBoardProg` call.

CLI Equivalent `BestXBoardDefaultSet`

CLI Abbreviation `boarddefset`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

See also *“BestXBoardGet” on page 34*
“BestXBoardProg” on page 35
“BestXBoardRead” on page 36
“BestXBoardReset” on page 36
“BestXBoardSet” on page 37

BestXBoardGet

Call

```
bx_errtype BestXBoardGet(
      bx_handletype   handle,
      bx_boardtype    prop,
      bx_int32ptr     *val );
```

Description

Gets a generic board property from the host storage. To read the properties from the testcard, use `BestXBoardRead` instead.

CLI Equivalent

```
BestXBoardGet prop=<prop>
```

CLI Abbreviation

```
boardget prop=<prop>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

prop Board property to be obtained; see *“bx_boardtype” on page 160*.

Output Parameters

val Value of the board property; see *“bx_boardtype” on page 160*.

See also

“BestXBoardDefaultSet” on page 33
“BestXBoardProg” on page 35
“BestXBoardRead” on page 36
“BestXBoardReset” on page 36
“BestXBoardSet” on page 37

BestXBoardProg

Call `bx_errtype BestXBoardProg( bx_handletype handle );`

Description Writes the board properties from the host storage to the testcard.

CLI Equivalent `BestXBoardProg`

CLI Abbreviation `boardprog`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle	Session identification.
prop	Board property; see *“bx_boardtype” on page 160*.

Output Parameters

value	Value of the board property; see *“bx_boardtype” on page 160*.

See also

“BestXBoardGet” on page 34
“BestXBoardDefaultSet” on page 33
“BestXBoardRead” on page 36
“BestXBoardReset” on page 36
“BestXBoardSet” on page 37

BestXBoardRead

Call `bx_errtype BestXBoardRead( bx_handletype handle );`

Description Reads board properties from the testcard and writes these properties to the host storage.

CLI Equivalent `BestXBoardRead`

CLI Abbreviation `boardread`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

prop Board property; see *"bx_boardtype" on page 160*.

value Value of the board property; see *"bx_boardtype" on page 160*.

See also *"BestXBoardGet" on page 34*
"BestXBoardDefaultSet" on page 33
"BestXBoardProg" on page 35
"BestXBoardReset" on page 36
"BestXBoardSet" on page 37

BestXBoardReset

Call `bx_errtype BestXBoardReset( bx_handletype handle );`

Description Resets the testcard.

CLI Equivalent `BestXBoardReset`

CLI Abbreviation `boardreset`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also *"BestXBoardGet" on page 34*
"BestXBoardDefaultSet" on page 33
"BestXBoardProg" on page 35
"BestXBoardRead" on page 36
"BestXBoardSet" on page 37

BestXBoardSet

Call

```
bx_errtype BestXBoardSet(
      bx_handletype      handle,
      bx_boardproptype   prop,
      bx_int32           val );
```

Description Sets a generic board property on the controlling host. To write the properties to the testcard, use `BestXBoardProg` instead.

CLI Equivalent `BestXBoardSet prop=<boardprop> val=<value>`

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

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

prop Property to be set; see *“bx_boardtype” on page 160*.

val Value to which the property is set; see *“bx_boardtype” on page 160*.

See also *“BestXBoardGet” on page 34*
“BestXBoardDefaultSet” on page 33
“BestXBoardProg” on page 35
“BestXBoardRead” on page 36
“BestXBoardReset” on page 36

BestXPUProg

Call `bx_errtype BestXPUProg( bx_handletype handle );`

Description Writes the current settings of the configuration space including decoder settings to the non-volatile memory. The new settings are activated after the next power up.

CLI Equivalent `BestXPUProg`

CLI Abbreviation `puprog`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

See also *“BestXAllDefaultSet” on page 33*

Power Management Event Functions

The following functions are used to supervise power management events:

Function	Action
"BestXPMEWrite" on page 39	Defines the status of the power management enable pin.
"BestXPMERead" on page 38	Determines if a power management event has occurred.

BestXPMERead

Call

```
bx_errtype BestXPMERead(
      bx_handletype     handle,
      bx_int32          *value );
```

Description Reads the status of the Power Management Enable pin on the testcard.

CLI Equivalent `BestXPMERead`

CLI Abbreviation `pmewrite`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

Output Parameters **value** Value of the pin:

- 0 – no event
- 1 – event

See also *"BestXPMEWrite" on page 39*

BestXPMEWrite

Call

```
bx_errtype BestXPMEWrite(
      bx_handletype      handle,
      bx_int32           value );
```

Description Writes the value of the Power Management Enable pin to the testcard.

CLI Equivalent `BestXPMEWrite value=<value>`

CLI Abbreviation `pmewrite val=<val>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

value Value of the pin:

- 0 – clears the event
- 1 – sets the event

See also *“BestXPMERead” on page 38*

Interrupt Generation Function

The following function is used to generate a PCI-X interrupt:

Function	Action
"BestXInterruptGenerate" on page 40	Generates a PCI-X interrupt.

BestXInterruptGenerate

Call

```
bx_errtype BestXInterruptGenerate(
      bx_handletype   handle,
      bx_int32        intr );
```

Description

Sets the specified PCI-X interrupt request pins `INTA#` … `INTD#`.

To reset the interrupt, clear the corresponding status bit in the status register with *"BestXStatusClear" on page 30*.

CLI Equivalent

```
BestXInterruptGenerate intr=<intr>
```

CLI Abbreviation

```
interruptgenerate intr=<intr>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

interrupt Interrupt to be generated. See the table below.

Multiple interrupts can be set by OR-combining the interrupt values, for example, `interrupt=BX_INTA | BX_INTC`.

Value	CLI Abbreviation	Description
BX_INTA … BX_INTD	inta … intd	PCI-X Interrupt Request A … D (INTA# … INTD#)

See also

–

Standard Analysis Functions

The application programming interface of the testcard provides functions to set up the protocol observer.

These functions can be used to mask individual protocol rules and to report all the protocol rule violations that occurred.

Protocol Observer Functions

The following functions are used for programming the protocol observer:

Function	Action
"BestXObsMaskProg" on page 43	Sets an individual error mask bit.
"BestXObsMaskRead" on page 44	Reads an individual error mask bit.
"BestXObsStatusRead" on page 47	Returns all errors found in the error registers in a text string.
"BestXObsBitPosGet" on page 42	Returns the bit position of a specific rule.
"BestXObsRuleStringGet" on page 46	Returns the protocol rule text for a specific error.
"BestXObsRuleGet" on page 45	Returns the protocol rule identifier for a specific error.

BestXObsBitPosGet

Call

```
bx_errtype BestXObsBitPosGet(
      bx_handletype    handle
      bx_obsruletype   rule,
      bx_int32         *bitpos );
```

Description Gets the bit position of the specified observer rule.

CLI Equivalent `BestXObsBitPosGet rule=<rule>`

CLI Abbreviation `obsbitposget rule=<rule>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

rule Identifier of the observer rule; see *“bx_obsruletype” on page 177*.

Output Parameters **bitpos** Bit position in one of the error registers that specifies the rule. The range is 0 … 52.

See also *“BestXObsRuleGet” on page 45*

BestXObsMaskProg

Call

```
bx_errtype BestXObsMaskProg(
      bx_handletype    handle,
      bx_obsruletype   rule,
      bx_int32         val );
```

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

NOTE Violations of masked rules are not totally ignored; the rule violation is indicated in the accumulated error register.

CLI Equivalent `BestXObsMaskProg rule=<rule> val=<val>`

CLI Abbreviation `obsmaskprog rule=<rule> val=<val`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

rule Protocol rule to be masked; see *“bx_obsruletype” on page 177*.

value Value to be entered in the mask:

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

See also *“BestXObsMaskRead” on page 44*

BestXObsMaskRead

Call

```
bx_errtype BestXObsMaskRead(
      bx_handletype    handle,
      bx_obsruletype   rule,
      bx_int32         *val );
```

Description Reads the rule mask of the specified rule from the testcard.

CLI Equivalent `BestXObsMaskRead rule=<rule>`

CLI Abbreviation `obsmaskread rule=<rule>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

rule Protocol rule identifier; see *"bx_obsruletype" on page 177*.

Output Parameters **val** Value of the mask bit:

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

See also *"BestXObsMaskProg" on page 43*

BestXObsRuleGet

Call

```
bx_errtype BestXObsRuleGet(
      bx_handletype    handle
      bx_int32         bitpos,
      bx_obsruletype   *rule );
```

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

CLI Equivalent `BestXObsRuleGet bitpos=<bitpos>`

CLI Abbreviation `obsruleget bitpos=<bitpos>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

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

Output Parameters

rule Identifier of the observer rule; see *“bx_obsruletype” on page 177*.

See also *“BestXObsBitPosGet” on page 42*

BestXObsRuleStringGet

Call

```
bx_errtype BestXObsRuleStringGet(
      bx_handletype    handle,
      bx_obsruletype   rule,
      bx_charptrtype   *errortext );
```

Description

Returns a text string that describes the rule as specified by the bit position of rule violation in one of the observer error registers (first error or accumulated error registers).

You can query the bit positions of violated rule(s) by using the `BestXObsBitPosGet` call.

Example:

If rule 13 violation is indicated by one of the error registers and you call this function with this value, the function will return the rule: "IRDY# must be asserted two clocks after the attribute phase".

CLI Equivalent

```
BestXObsRuleStringGet rule=<rule>
```

CLI Abbreviation

```
obsrulestringget rule=<rule>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

rule Bit position that specifies the rule; see *"bx_obsruletype" on page 177*.

Output Parameters

errortext Text string that describes the rule.

See also

"BestXObsRuleGet" on page 45

BestXObsStatusRead

Call

```
bx_errtype BestXObsStatusRead(
      bx_handletype    handle,
      bx_charptrtype   *errortext );
```

Description Returns a text string containing all errors that were found in the first error register and the accumulated error registers. A combined error report string will be generated for this purpose.

CLI Equivalent `BestXObsStatusRead`

CLI Abbreviation `obsstatusread`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

Output Parameters **errortext** Error report string that contains all rule violations. For the list of rules, see *"bx_obsruletype" on page 177*.

See also –

Exerciser Functions

The PCI-X exerciser functions are divided as follows:

Exerciser Functions	Action
Generic Exerciser Functions	Prepare for exerciser programming.
Requester-Initiator Programming Functions	Set and get requester-initiator properties (generic, block and behavior properties).
Requester-Target Programming Functions	Set and get requester-target properties (generic and behavior properties).
Completer-Target Programming Functions	Set and get completer-target properties (generic and behavior properties).
Completer-Initiator Programming Functions	Set and get completer-initiator properties (generic and behavior properties).
Configuration Space Programming Functions	Set and get configuration space register and configuration space mask register.
Decoder Programming Functions	Set and get target and split decoder properties.
Expansion ROM Programming Functions	Write and read data to the expansion ROM.
Data Memory Functions	Write and read the data memory.
Host Access Functions	Set and get registers on the PCI-X bus.

Generic Exerciser Functions

The following functions are used to prepare for exerciser programming:

Function	Action
"BestXExerciserDefaultSet" on page 52	Sets all exerciser properties to default values.
"BestXExerciserGenDefaultSet" on page 52	Sets the exerciser generic properties to default values.
"BestXExerciserGenSet" on page 54	Sets an exerciser generic property.
"BestXExerciserGenGet" on page 53	Gets an exerciser generic property.
"BestXExerciserProg" on page 55	Loads exerciser settings and properties to the testcard.
"BestXExerciserRun" on page 57	Resets all initiator and target behavior counters and starts requester-initiator transactions.
"BestXExerciserRead" on page 55	Reads all Exerciser properties from the testcard to the host storage.
"BestXExerciserStop" on page 57	Stops the requester-initiator.
"BestXExerciserBreak" on page 51	Resets the complete exerciser and leaves the bus in an clear state.
"BestXExerciserPause" on page 54	Completes the current transaction and halts the requester-initiator.
"BestXExerciserContinue" on page 51	Runs requester-initiator transactions without resetting counters.
"BestXExerciserReset" on page 56	Resets the complete exerciser and leaves the bus in an unclear state.

BestXExerciserBreak

Call `bx_errtype BestXExerciserBreak( bx_handletype handle );`

Description Completes the current transaction, disables the target and resets the complete exerciser afterwards. The function leaves the bus in a clear state.

CLI Equivalent `BestXExerciserBreak`

CLI Abbreviation `ebreak`

Return Value Error code; see *“bx_errtype” on page 176.*

Input Parameters **handle** Session identification.

See also *“BestXExerciserReset” on page 56*

BestXExerciserContinue

Call `bx_errtype BestXExerciserContinue( bx_handletype handle );`

Description Continues requester-initiator transactions without resetting the behavior counters.

CLI Equivalent `BestXExerciserContinue`

CLI Abbreviation `econtinue`

Return Value Error code; see *“bx_errtype” on page 176.*

Input Parameters **handle** Session identification.

See also *“BestXExerciserReset” on page 56*

BestXExerciserDefaultSet

Call `bx_errtype BestXExerciserDefaultSet( bx_handletype handle );`

Description Sets all exerciser properties on the host to default values. This includes:

- generic properties
- memories (block and behavior properties)
- decoders
- split-response condition properties
- the configuration space

CLI Equivalent `BestXExerciserDefaultSet`

CLI Abbreviation `edefset`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also –

BestXExerciserGenDefaultSet

Call `bx_errtype BestXExerciserGenDefaultSet( bx_handletype handle );`

Description Sets all generic exerciser properties to default values. For default values, see *"bx_egentype" on page 170*.

CLI Equivalent `BestXExerciserGenDefaultSet`

CLI Abbreviation `egendefset`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also *"BestXExerciserGenSet" on page 54*
"BestXExerciserGenGet" on page 53

BestXExerciserGenGet

Call

```
bx_errtype BestXExerciserGenGet(
      bx_handletype   handle,
      bx_egentype     prop,
      bx_int32        *val );
```

Description Gets a generic exerciser property from the host storage.

CLI Equivalent `BestXExerciserGenGet prop=<prop>`

CLI Abbreviation `egenget prop=<prop>`

Return Value Error code; see *"bx_errtype" on page 176.*

Input Parameters **handle** Session identification.

prop Exerciser property to be obtained; see *"bx_egentype" on page 170.*

Output Parameters **val** Value of the exerciser property; see *"bx_egentype" on page 170.*

See also *"BestXExerciserGenSet" on page 54*
"BestXExerciserGenDefaultSet" on page 52
"BestXExerciserProg" on page 55

BestXExerciserGenSet

Call

```
bx_errtype BestXExerciserGenSet(
      bx_handletype   handle,
      bx_egentype     prop,
      bx_int32        val );
```

Description Sets a generic exerciser property on the host storage. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent `BestXExerciserGenSet prop=<prop> val=<val>`

CLI Abbreviation `egenset prop=<prop> val=<val>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

prop Exerciser property to be set; see *“bx_egentype” on page 170*.

val Value to which the exerciser property is set; see *“bx_egentype” on page 170*.

See also *“BestXExerciserProg” on page 55*
“BestXExerciserGenGet” on page 53
“BestXExerciserGenDefaultSet” on page 52

BestXExerciserPause

Call

```
bx_errtype BestXExerciserPause( bx_handletype handle );
```

Description Completes the current transaction and halts the requester-initiator. The behavior counters are not reset.

CLI Equivalent `BestXExerciserPause`

CLI Abbreviation `epause`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

See also *“BestXExerciserContinue” on page 51*

BestXExerciserProg

Call

```
bx_errtype BestXExerciserProg( bx_handletype handle );
```

Description Writes all previously programmed exerciser properties to the testcard.

CLI Equivalent `BestXExerciserProg`

CLI Abbreviation `eprog`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

See also –

BestXExerciserRead

Call

```
bx_errtype BestXExerciserRead(
      bx_handletype   handle,
      bx_int32        option )
```

Description Reads all Exerciser properties from the testcard to the host storage. This function is used to observe changes in the testcard setting during its configuration.

CLI Equivalent `BestXExerciserRead option=<option>`

CLI Abbreviation `eread option=<option>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

option Determines whether all memories (1) or just the generic properties (0) are to be read.

See also *“BestXExerciserProg” on page 55*

BestXExerciserReset

Call

```
bx_errtype BestXExerciserReset( bx_handletype handle );
```

Description

Resets all bus state machines (initiator and target) and clears the requester-initiator intention.

CAUTION

This function call might lead to a behavior that is not protocol conform. All state machines are reset regardless of their current state. Hence, the bus state will be unclear. This call is only useful if a bus is hung.

CLI Equivalent

```
BestXExerciserReset
```

CLI Abbreviation

```
ereset
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle	Session identification.

See also

"BestXExerciserBreak" on page 51

BestXExerciserRun

Call `bx_errtype BestXExerciserRun( bx_handletype handle );`

Description Resets all initiator and target behavior counters and starts requester-initiator transactions.

CLI Equivalent `BestXExerciserRun`

CLI Abbreviation `erun`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

See also *“BestXExerciserStop” on page 57*
“BestXExerciserContinue” on page 51
“BestXExerciserPause” on page 54
“BestXRIGenSet” on page 67
“bx_rigentype” on page 187

BestXExerciserStop

Call `bx_errtype BestXExerciserStop( bx_handletype handle );`

Description Completes the current transaction and then stops the requester-initiator.

CLI Equivalent `BestXExerciserStop`

CLI Abbreviation `estop`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

See also *“BestXExerciserRun” on page 57*
“BestXExerciserContinue” on page 51
“BestXExerciserPause” on page 54

Requester-Initiator Programming Functions

The following functions are used to read and write generic properties, block properties and behavior properties for the requester-initiator:

Function	Action
“BestXRIGenDefaultSet” on page 66	Sets all generic requester-initiator properties to default values.
“BestXRIGenSet” on page 67	Sets generic requester-initiator properties.
“BestXRIGenGet” on page 66	Gets generic requester-initiator properties.
“BestXRIBlockMemInit” on page 64	Initializes the requester-initiator block memory.
“BestXRIBlockDefaultSet” on page 63	Sets all requester-initiator block properties to default values.
“BestXRIBlockSet” on page 65	Sets requester-initiator block properties.
“BestXRIBlockGet” on page 64	Gets requester-initiator block properties.
“BestXRIBehDefaultSet” on page 59	Sets all requester-initiator behavior properties to default values.
“BestXRIBehSet” on page 62	Sets all requester-initiator behavior properties.
“BestXRIBehGet” on page 60	Gets all requester-initiator behavior properties.

BestXRIBehDefaultSet

Call

```
bx_errtype BestXRIBehDefaultSet(
      bx_handletype   handle,
      bx_int32        offset );
```

Description

Sets all requester-initiator behavior properties on one line of the requester-initiator behavior memory on the host to default values. For default values, see *"bx_ribehtype" on page 182*.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXRIBehDefaultSet offset=<offset>
```

CLI Abbreviations

```
ribehdefset offset=<offset>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

offset Memory line that is set to default values (0 … 255).

See also

"BestXRIBehSet" on page 62
"BestXRIBehGet" on page 60
"BestXExerciserProg" on page 55

BestXRIBehGet

Call

```
bx_errtype BestXRIBehGet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_ribehtype    prop,
      bx_int32        *val );
```

Description Gets a requester-initiator behavior property from one line of the requester-initiator behavior memory on the host.

CLI Equivalent `BestXRIBehGet offset=<offset> prop=<prop>`

CLI Abbreviations `ribehget offset=<offset> prop=<prop>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

offset Line of the requester-initiator behavior memory (0 … 255).

prop Requester-initiator behavior property to be obtained; see *“bx_ribehtype” on page 182*.

Output Parameters

val Value of the requester-initiator behavior property; see *“bx_ribehtype” on page 182*.

See also

“BestXRIBehDefaultSet” on page 59
“BestXRIBehSet” on page 62
“BestXRIBlockGet” on page 64

BestXRIBehMemInit

Call `bx_errtype BestXRIBehMemInit( bx_handletype handle );`

Description Sets default values to the entire requester-initiator behavior memory on the host (256 lines). For the default values, see *"bx_ribehtype" on page 182*.

CLI Equivalent `BestXRIBehMemInit`

CLI Abbreviation `ribehmeminit`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also *"BestXRIBlockMemInit" on page 64*
"BestXCIBehMemInit" on page 84
"BestXRTBehMemInit" on page 70
"BestXCTBehMemInit" on page 77

BestXRIBehSet

Call

```
bx_errtype BestXRIBehSet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_ribehtype    prop,
      bx_int32        val );
```

Description Sets a requester-initiator behavior property on one line of the requester-initiator behavior memory on the host. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent `BestXRIBehSet offset=<offset> prop=<prop> val=<val>`

CLI Abbreviations `ribehset offset=<offset> prop=<prop> val=<val>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

offset Line of the requester-initiator behavior memory (0 … 255).

prop Requester-initiator behavior property to be set; see *“bx_ribehtype” on page 182*.

val Value of the requester-initiator behavior property to be set; see *“bx_ribehtype” on page 182*.

See also *“BestXRIBehDefaultSet” on page 59*
“BestXRIBehGet” on page 60
“BestXRIBlockSet” on page 65

BestXRIBlockDefaultSet

Call

```
bx_errtype BestXRIBlockDefaultSet(
      bx_handletype   handle,
      bx_int32        offset );
```

Description

Sets all requester-initiator block properties on one line of the requester-initiator block memory on the host to default values.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXRIBlockDefaultSet offset=<offset>
```

CLI Abbreviations

```
riblkdefset offset=<offset>
```

Return Value

Error code; see *“bx_errtype” on page 176.*

Input Parameters

handle Session identification.

offset Line that is set to default values (0 … 255).

See also

“BestXRIBlockSet” on page 65
“BestXRIBlockGet” on page 64
“BestXRIBehDefaultSet” on page 59

BestXRIBlockGet

Call

```
bx_errtype BestXRIBlockGet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_riblktype    prop,
      bx_int32        *val );
```

Description Gets a requester-initiator block property from one line of the requester-initiator block memory on the host.

CLI Equivalent `BestXRIBlockGet offset=<offset> prop=<prop>`

CLI Abbreviation `riblkget offset=<offset> prop=<prop>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

offset Line of the requester-initiator block memory (0 … 255).

prop Requester-initiator block property to be obtained; see *“bx_riblktype” on page 184*.

Output Parameters

val Value of the requester-initiator block property; see *“bx_riblktype” on page 184*.

See also

“BestXRIBlockDefaultSet” on page 63
“BestXRIBlockSet” on page 65
“BestXRIBehGet” on page 60

BestXRIBlockMemInit

Call `bx_errtype BestXRIBlockMemInit( bx_handletype handle );`

Description Sets default values to the entire requester-initiator block memory on the host (256 lines). For the default values, see *“bx_riblktype” on page 184*.

CLI Equivalent `BestXRIBlockMemInit`

CLI Abbreviation `riblkmeminit`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

See also *“BestXRIBehMemInit” on page 61*

BestXRIBlockSet

Call

```
bx_errtype BestXRIBlockSet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_riblktype    prop,
      bx_int32        val );
```

Description

Sets a requester-initiator block property on one line of the requester-initiator block memory on the host. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXRIBlockSet offset=<offset> prop=<prop> val=<val>
```

CLI Abbreviation

```
riblkset offset=<offset> prop=<prop> val=<val>
```

Return Value

Error code; see *“bx_errtype” on page 176.*

Input Parameters

handle Session identification.

offset Line of the requester-initiator block memory (0 … 255).

prop Requester-initiator block property to be set; see *“bx_riblktype” on page 184.*

val Value of the requester-initiator block property to be set; see *“bx_riblktype” on page 184.*

See also

“BestXRIBlockDefaultSet” on page 63
“BestXRIBlockGet” on page 64
“BestXRIBehSet” on page 62
“BestXExerciserProg” on page 55

BestXRIGenDefaultSet

Call

```
bx_errtype BestXRIGenDefaultSet( bx_handletype handle );
```

Description

Sets all generic requester-initiator properties on the host to their default values. For a list of these properties, see *“bx_rigentype” on page 187*.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXRIGenDefaultSet
```

CLI Abbreviation

```
rigendefset
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

See also

“BestXRIGenSet” on page 67
“BestXRIGenGet” on page 66

BestXRIGenGet

Call

```
bx_errtype BestXRIGenGet(
      bx_handletype   handle,
      bx_rigentype    prop,
      bx_int32        *val );
```

Description

Gets the value of a generic requester-initiator property from the host.

CLI Equivalent

```
BestXRIGenGet prop=<prop>
```

CLI Abbreviation

```
rigenget prop=<prop>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

prop Property to be obtained; see *“bx_rigentype” on page 187*.

Output Parameters

val Property value; see *“bx_rigentype” on page 187*.

See also

“BestXRIGenDefaultSet” on page 66
“BestXRIGenSet” on page 67

BestXRIGenSet

Call

```
bx_errtype BestXRIGenSet(
      bx_handletype   handle,
      bx_rigentype    prop,
      bx_int32        val );
```

Description Sets the value of a generic requester-initiator property on the host. Generic requester-initiator properties are valid during a requester-initiator run. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent `BestXRIGenSet prop=<prop> val=<val>`

CLI Abbreviation `rigenset prop=<prop> val=<val>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

prop Property to be set; see *"bx_rigentype" on page 187*.

val Value to which the property is set; see *"bx_rigentype" on page 187*.

See also *"BestXRIGenDefaultSet" on page 66*
"BestXRIGenGet" on page 66
"BestXExerciserProg" on page 55

Requester-Target Programming Functions

The following functions are used to read and write generic properties and behavior properties for the requester-target:

Function	**Action**
“BestXRTGenDefaultSet” on page 72	Sets all generic requester-target properties to default values.
“BestXRTGenSet” on page 73	Sets generic requester-target properties.
“BestXRTGenGet” on page 72	Gets generic requester-target properties.
“BestXRTBehMemInit” on page 70	Initializes the requester-target behavior memory.
“BestXRTBehDefaultSet” on page 69	Sets all requester-target behavior properties on one line of the behavior memory to default values.
“BestXRTBehSet” on page 71	Sets a requester-target behavior property on one line of the behavior memory.
“BestXRTBehGet” on page 70	Gets a requester-target behavior property from one line of the behavior memory.

BestXRTBehDefaultSet

Call

```
bx_errtype BestXRTBehDefaultSet(
      bx_handletype   handle,
      bx_int32        offset );
```

Description

Sets all requester-target behavior properties on one line of the requester-target behavior memory on the host to default values. For default values, see *"bx_rtbehtype" on page 188*.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXRTBehDefaultSet offset=<offset>
```

CLI Abbreviation

```
rtbehdefset offset=<offset>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

offset Behavior line that is set to default values (0 … 255).

See also

"BestXRTBehSet" on page 71
"BestXRTBehGet" on page 70
"BestXExerciserProg" on page 55

BestXRTBehGet

Call

```
bx_errtype BestXRTBehGet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_rtbehtype    prop,
      bx_int32        *val );
```

Description

Gets a requester-target behavior property from one line of the requester-target behavior memory on the host.

CLI Equivalent

`BestXRTBehGet offset=<offset> prop=<prop>`

CLI Abbreviation

`rtbehget offset=<offset> prop=<prop>`

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

offset Line of the requester-target behavior memory (0 … 255).

prop Requester-target behavior property to be obtained; see *"bx_rtbehtype" on page 188*.

Output Parameters

val Value of the requester-target behavior property; see *"bx_rtbehtype" on page 188*.

See also

"BestXRTBehDefaultSet" on page 69
"BestXRTBehSet" on page 71

BestXRTBehMemInit

Call

```
bx_errtype BestXRTBehMemInit( bx_handletype handle );
```

Description

Sets default values to the entire requester-target behavior memory on the host (256 lines). For the default values, see *"bx_rtbehtype" on page 188*.

CLI Equivalent

`BestXRTBehMemInit`

CLI Abbreviation

`rtbehmeminit`

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

See also

"BestXRIBehMemInit" on page 61
"BestXCIBehMemInit" on page 84
"BestXCTBehMemInit" on page 77

BestXRTBehSet

Call

```
bx_errtype BestXRTBehSet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_rtbehtype    prop,
      bx_int32        val );
```

Description

Sets a requester-target behavior property on one line of the requester-target behavior memory on the host. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXRTBehSet offset=<offset> prop=<prop> val=<val>
```

CLI Abbreviation

```
rtbehset offset=<offset> prop=<prop> val=<val>
```

Return Value

Error code; see *“bx_errtype” on page 176.*

Input Parameters

handle Session identification.

offset Line of the requester-target behavior memory (0 … 255).

prop Requester-target behavior property to be set; see *“bx_rtbehtype” on page 188.*

val Value of the requester-target behavior property to be set; see *“bx_rtbehtype” on page 188.*

See also

“BestXRTBehDefaultSet” on page 69
“BestXRTBehGet” on page 70
“BestXExerciserProg” on page 55

BestXRTGenDefaultSet

Call

```
bx_errtype BestXRTGenDefaultSet ( bx_handletype handle );
```

Description

Sets the generic requester-target properties on the host to their default values. For a list of these properties, see *"bx_rtgentype" on page 189*.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXRTGenDefaultSet
```

CLI Abbreviation

```
rtgendefset
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

See also

"BestXRTGenSet" on page 73
"BestXRTGenGet" on page 72
"BestXExerciserProg" on page 55

BestXRTGenGet

Call

```
bx_errtype BestXRTGenGet(
      bx_handletype   handle,
      bx_rtgentype    prop,
      bx_int32        *val );
```

Description

Gets a generic requester-target property from the host.

CLI Equivalent

```
BestXRTGenGet prop=<prop>
```

CLI Abbreviation

```
rtgenget prop=<prop>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

prop Property to be obtained; see *"bx_rtgentype" on page 189*.

Output Parameters

val Property value; see *"bx_rtgentype" on page 189*.

See also

"BestXRTGenDefaultSet" on page 72
"BestXRTGenSet" on page 73

BestXRTGenSet

Call

```
bx_errtype BestXRTGenSet(
      bx_handletype   handle,
      bx_rtgentype    prop,
      bx_int32        val );
```

Description

Sets a generic requester-target property on the host. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXRTGenSet prop=<prop> val=<val>
```

CLI Abbreviation

```
rtgenset prop=<prop> val=<val>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

prop Property to be set; see *"bx_rtgentype" on page 189*.

val Value to which the property is set; see *"bx_rtgentype" on page 189*.

See also

"BestXRTGenDefaultSet" on page 72
"BestXRTGenGet" on page 72
"BestXExerciserProg" on page 55

Completer-Target Programming Functions

The following functions are used to read and write generic properties and behavior properties for the completer-target:

Function	Action
"BestXCTGenDefaultSet" on page 79	Sets all generic completer-target properties to default values.
"BestXCTGenSet" on page 80	Sets generic completer-target properties.
"BestXCTGenGet" on page 79	Gets generic completer-target properties.
"BestXCTBehMemInit" on page 77	Initializes the completer-target behavior memory.
"BestXCTBehDefaultSet" on page 75	Sets all completer-target behavior properties on one line of the behavior memory to default values.
"BestXCTBehSet" on page 78	Sets a completer-target behavior property on one line of the behavior memory.
"BestXCTBehGet" on page 76	Gets a completer-target behavior property from one line of the behavior memory.

BestXCTBehDefaultSet

Call

```
bx_errtype BestXCTBehDefaultSet(
      bx_handletype   handle,
      bx_int32        offset );
```

Description

Sets all completer-target behavior properties on one line of the completer-target behavior memory on the host to default values. For default values, see *“bx_ctbehtype” on page 163*.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXCTBehDefaultSet offset=<offset>
```

CLI Abbreviation

```
ctbehdefset offset=<offset>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

offset Line that is set to default values (0 … 255).

See also

“BestXCTBehSet” on page 78
“BestXCTBehGet” on page 76
“BestXExerciserProg” on page 55

BestXCTBehGet

Call

```
bx_errtype BestXCTBehGet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_ctbehtype    prop,
      bx_int32        *val );
```

Description Gets a completer-target behavior property from one line of the completer-target behavior memory on the host.

CLI Equivalent `BestXCTBehGet offset=<offset> prop=<prop>`

CLI Abbreviation `ctbehget offset=<offset> prop=<prop>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

offset Line of the completer-target behavior memory (0 … 255).

prop Completer-target behavior property to be obtained; see *“bx_ctbehtype” on page 163*.

Output Parameters

val Value of the completer-target behavior property; see *“bx_ctbehtype” on page 163*.

See also

“BestXCTBehDefaultSet” on page 75
“BestXCTBehSet” on page 78

BestXCTBehMemInit

Call `bx_errtype BestXCTBehMemInit( bx_handletype handle );`

Description Sets default values to the entire completer-target behavior memory on the host (256 lines). For the default values, see *"bx_ctbehtype" on page 163*.

CLI Equivalent `BestXCTBehMemInit`

CLI Abbreviation `ctbehmeminit`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also *"BestXRIBehMemInit" on page 61*
"BestXCIBehMemInit" on page 84
"BestXRTBehMemInit" on page 70

BestXCTBehSet

Call

```
bx_errtype BestXCTBehSet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_ctbehtype    prop,
      bx_int32        val );
```

Description

Sets a completer-target behavior property on one line of the completer-target behavior memory on the host. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXCTBehSet offset=<offset> prop=<prop> val=<val>
```

CLI Abbreviation

```
ctbehset offset=<offset> prop=<prop> val=<val>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

offset Line of the completer-target behavior memory (0 … 255).

prop Completer-target behavior property to be set; see *“bx_ctbehtype” on page 163*.

val Value of the completer-target behavior property to be set; see *“bx_ctbehtype” on page 163*.

See also

“BestXCTBehDefaultSet” on page 75
“BestXCTBehGet” on page 76
“BestXExerciserProg” on page 55

BestXCTGenDefaultSet

Call

```
bx_errtype BestXCTGenDefaultSet( bx_handletype handle );
```

Description

Sets the generic completer-target properties on the host to their default values. For a list of these properties, see *"bx_ctgentype" on page 165*.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXCTGenDefaultSet
```

CLI Abbreviation

```
ctgendefset
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

See also

"BestXCTGenSet" on page 80
"BestXCTGenGet" on page 79
"BestXExerciserProg" on page 55

BestXCTGenGet

Call

```
bx_errtype BestXCTGenGet(
      bx_handletype   handle,
      bx_cigentype    prop,
      bx_int32        *val );
```

Description

Gets a generic completer-target property from the host.

CLI Equivalent

```
BestXCTGenGet prop=<prop>
```

CLI Abbreviation

```
ctgenget prop=<prop>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

prop Property to be obtained; see *"bx_ctgentype" on page 165*.

Output Parameters

val Property value; see *"bx_ctgentype" on page 165*.

See also

"BestXCTGenDefaultSet" on page 79
"BestXCTGenSet" on page 80

BestXCTGenSet

Call

```
bx_errtype BestXCTGenSet(
      bx_handletype   handle,
      bx_ctgentype    prop,
      bx_int32        val );
```

Description Sets a generic completer-target property on the host. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent `BestXCTGenSet prop=<prop> val=<val>`

CLI Abbreviation `ctgenset prop=<prop> val=<val>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

prop Property to be set; see *“bx_ctgentype” on page 165*.

val Value to which the property is set; see *“bx_ctgentype” on page 165*.

See also *“BestXCTGenDefaultSet” on page 79*
“BestXCTGenGet” on page 79
“BestXExerciserProg” on page 55

Completer-Initiator Programming Functions

The following functions are used to read and write generic properties and behavior properties for the completer-initiator:

Function	Action
"BestXCIGenDefaultSet" on page 86	Sets all generic completer-initiator properties to default values.
"BestXCIGenSet" on page 87	Sets generic completer-initiator properties.
"BestXCIGenGet" on page 86	Gets generic completer-initiator properties.
"BestXCIBehMemInit" on page 84	Initializes the completer-initiator behavior memory.
"BestXCIBehDefaultSet" on page 82	Sets all completer-initiator behavior properties to default values.
"BestXCIBehSet" on page 85	Sets a completer-initiator behavior property on one line of the behavior memory.
"BestXCIBehGet" on page 83	Gets a completer-initiator behavior property from one line of the behavior memory.

BestXCIBehDefaultSet

Call

```
bx_errtype BestXCIBehDefaultSet(
      bx_handletype   handle,
      bx_int32        offset );
```

Description

Sets all completer-initiator properties on one line of the completer-initiator behavior memory on the host to default values. For default values, see *"bx_cibehtype" on page 161*.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXCIBehDefaultSet offset=<offset>
```

CLI Abbreviation

```
cibehdefset offset=<offset>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

offset Line that is set to default values (0 … 255).

See also

"BestXCIBehSet" on page 85
"BestXCIBehGet" on page 83
"BestXExerciserProg" on page 55

BestXCIBehGet

Call

```
bx_errtype BestXCIBehGet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_cibehtype    prop,
      bx_int32        *val );
```

Description Gets a completer-initiator behavior property from one line of the completer-initiator behavior memory on the host.

CLI Equivalent `BestXCIBehGet offset=<offset> prop=<prop>`

CLI Abbreviation `cibehget offset=<offset> prop=<prop>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

offset Line of the completer-initiator behavior memory (0 … 255).

prop Completer-initiator behavior property to be obtained; see *“bx_cibehtype” on page 161*.

Output Parameters

val Value of the completer-initiator behavior property; see *“bx_cibehtype” on page 161*.

See also

“BestXCIBehDefaultSet” on page 82
“BestXCIBehSet” on page 85

BestXCIBehMemInit

Call `bx_errtype BestXCIBehMemInit( bx_handletype handle );`

Description Sets default values to the entire completer-initiator behavior memory on the host (256 lines). For the default values, see *“bx_cibehtype” on page 161*.

CLI Equivalent `BestXCIBehMemInit`

CLI Abbreviation `cibehmeminit`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

See also *“BestXRIBehMemInit” on page 61*
“BestXRTBehMemInit” on page 70
“BestXCTBehMemInit” on page 77

BestXCIBehSet

Call

```
bx_errtype BestXCIBehSet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_cibehtype    prop,
      bx_int32        val );
```

Description

Sets a completer-initiator behavior property on one line of the completer-initiator behavior memory on the host. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

`BestXCIBehSet offset=<offset> prop=<prop> val=<val>`

CLI Abbreviation

`cibehset offset=<offset> prop=<prop> val=<val>`

Return Value

Error code; see *“bx_errtype” on page 176.*

Input Parameters

handle Session identification.

offset Line of the completer-initiator behavior memory (0 … 255).

prop Completer-initiator behavior property to be set; see *“bx_cibehtype” on page 161.*

val Value of the completer-initiator behavior property to be set; see *“bx_cibehtype” on page 161.*

See also

“BestXCIBehDefaultSet” on page 82
“BestXCIBehGet” on page 83
“BestXExerciserProg” on page 55

BestXCIGenDefaultSet

Call

```
bx_errtype BestXCIGenDefaultSet ( bx_handletype handle );
```

Description

Sets the generic completer-initiator properties on the host to their default values. For a list of these properties, see *"bx_cigentype" on page 163*.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXCIGenDefaultSet
```

CLI Abbreviation

```
cigendefset
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

See also

"BestXCIGenSet" on page 87
"BestXCIGenGet" on page 86
"BestXExerciserProg" on page 55

BestXCIGenGet

Call

```
bx_errtype BestXCIGenGet(
      bx_handletype   handle,
      bx_cigentype    prop,
      bx_int32        *val );
```

Description

Gets a generic completer-initiator property from the host.

CLI Equivalent

```
BestXCIGenGet prop=<prop>
```

CLI Abbreviation

```
cigenget prop=<prop>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

prop Property to be obtained; see *"bx_cigentype" on page 163*.

Output Parameters

val Property value; see *"bx_cigentype" on page 163*.

See also

"BestXCIGenDefaultSet" on page 86
"BestXCIGenSet" on page 87

BestXCIGenSet

Call

```
bx_errtype BestXCIGenSet(
      bx_handletype   handle,
      bx_cigentype    prop,
      bx_int32        val );
```

Description Sets a generic completer-initiator property on the host. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent `BestXCIGenSet prop=<prop> val=<val>`

CLI Abbreviation `cigenset prop=<prop> val=<val>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

prop Property to be set; see *“bx_cigentype” on page 163*.

val Value to which the property is set; see *“bx_cigentype” on page 163*.

See also

“BestXCIGenDefaultSet” on page 86
“BestXCIGenGet” on page 86
“BestXExerciserProg” on page 55

Configuration Space Programming Functions

The following functions are used to write and read configuration space and configuration mask registers:

Function	Action
“BestXConfRegSet” on page 91	Sets a configuration space register.
“BestXConfRegGet” on page 88	Gets a configuration space register.
“BestXConfRegMaskSet” on page 90	Sets a configuration mask register.
“BestXConfRegMaskGet” on page 89	Gets a configuration mask register.

BestXConfRegGet

Call

```
bx_errtype BestXConfRegGet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_int32        *val );
```

Description Gets a configuration space register from the host.

CLI Equivalent `BestXConfRegGet offset=<offset>`

CLI Abbreviation `confregget offset=<offset>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

offset Address offset in the configuration space. The offset needs to be DWORD-aligned. Valid values are `0x0` … `0x7c` (excluding the values `0x40` … `0x5c`).

Output Parameters **val** Value of the register (32-bit value).

See also *“BestXConfRegSet” on page 91*

BestXConfRegMaskGet

Call

```
bx_errtype BestXConfRegMaskGet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_int32        *val );
```

Description Gets the mask that defines which bits in a configuration space register on the host are set to read-only and which are set to read/write for configuration accesses.

CLI Equivalent `BestXConfRegMaskGet offset=<offset>`

CLI Abbreviation `confregmaskget offset=<offset>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

offset Address offset in the configuration space. The offset needs to be DWORD-aligned. Valid values are `0x0` … `0x7c` (excluding the values `0x40` … `0x5c`).

Output Parameters **val** 32-bit mask:

- 0 – read-only
- 1 – read/writeable

See also *“BestXConfRegMaskSet” on page 90*

BestXConfRegMaskSet

Call

```
bx_errtype BestXConfRegMaskSet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_int32        val );
```

Description Sets the mask that defines which bits in a configuration space register on the host are set to read-only and which to read/writeable from PCI-X. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent `BestXConfRegMaskSet offset=<offset> val=<val>`

CLI Abbreviation `confregmaskset offset=<offset> val=<val>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

offset Address offset in the configuration space. The offset needs to be DWORD-aligned. Valid values are `0x0` … `0x7c` (excluding the values `0x40` … `0x5c`).

val 32-bit mask:

- 0 – read-only
- 1 – read/writeable

See also *“BestXConfRegMaskGet” on page 89*
“BestXExerciserProg” on page 55

BestXConfRegSet

Call

```
bx_errtype BestXConfRegSet(
      bx_handletype   handle,
      bx_int32        offset,
      bx_int32        val );
```

Description

Sets a register of the configuration space on the host to the specified value. You can write either the entire value or only a part of it to the testcard by using the `BestXExerciserProg` call. Whether the entire value or a part of it is written to the testcard is defined by the property `BX_BOARD_RESPECTBIOS`; see *"bx_boardtype" on page 160*.

CAUTION

Do not set several address spaces to the same decoding location because the system under test can crash or may even be damaged.

To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXConfRegSet offset=<offset> val=<val>
```

CLI Abbreviation

```
confregset offset=<offset> val=<val>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

offset Address offset in the configuration space. The offset needs to be DWORD-aligned. Valid values are `0x0` … `0x7c` (excluding the values `0x40` … `0x5c`).

val Value to which the register is set (32-bit value).

See also

"BestXConfRegGet" on page 88
"BestXExerciserProg" on page 55

Decoder Programming Functions

The following functions are used to write and read target decoder and split decoder properties:

Function	Action
“BestXTDecoderDefaultSet” on page 94	Sets all target decoder properties to default values.
“BestXTDecoderAllSet” on page 93	Sets all target decoder properties.
“BestXTDecoderSet” on page 96	Sets a target decoder property.
“BestXTDecoderGet” on page 95	Gets a target decoder property.
“BestXCTSplitCondDefaultSet” on page 97	Sets all split decoder properties to default values.
“BestXCTSplitCondSet” on page 99	Sets a split decoder property.
“BestXCTSplitCondGet” on page 98	Gets a split decoder property.

BestXTDecoderAllSet

Call

```
bx_errtype BestXTDecoderAllSet(
          bx_handletype   handle,
          bx_dectype      dec,
          bx_int32        location,
          bx_int32        size,
          bx_int32        prefetch,
          bx_int32        baselo,
          bx_int32        basehi,
          bx_int32        resource,
          bx_int32        resbase,
          bx_int32        ressize,
          bx_int32        compare );
```

Description Sets all properties for a BAR decoder (BAR 0 … 5).

CLI Equivalent

```
BestXTDecoderAllSet dec=<dec> location=<location> size=<size>
prefetch=<prefetch> baselo=<baselo> basehi=<basehi>
resource=<resource> resbase=<resbase> ressize=<ressize>
compare=<compare>
```

CLI Abbreviation

```
tdecallset dec=<dec> location=<location> size=<size>
prefetch=<prefetch> baselo=<baselo> basehi=<basehi>
resource=<resource> resbase=<resbase> ressize=<ressize>
compare=<compare>
```

Return Value Error code; see *"bx_errtype" on page 176.*

Input Parameters

handle Session identification.

dec Target decoder to be set. Valid properties are `bar0` … `bar5`. See *"bx_dectype" on page 169.*

location Location for the requested address range to be set. See *"bx_decproptype" on page 168.*

size Decoder size to be set. See *"bx_decproptype" on page 168.*

prefetch Defines if the memory is prefetchable. See *"bx_decproptype" on page 168.*

baselo Lower 32 bits of the base address.

basehi Upper 32 bits of the base address.

resource Resource to be set. See *"bx_decproptype" on page 168.*

resbase Internal resource base address to be set. See *"bx_decproptype" on page 168*.

ressize Size of the internal resource to be set. See *"bx_decproptype" on page 168*.

compare Defines whether the data compare is switched on for this decoder. See *"bx_decproptype" on page 168*.

See also

"BestXTDecoderDefaultSet" on page 94
"BestXTDecoderGet" on page 95
"BestXTDecoderSet" on page 96
"BestXExerciserProg" on page 55

BestXTDecoderDefaultSet

Call

```
bx_errtype BestXTDecoderDefaultSet( bx_handletype handle );
```

Description

Sets all decoder properties for all target decoders on the host to default values.

Decoder	Defaults	Comment
BAR 0 and 1	- memory decoder with size 1 MB - prefetchable - resource base 0	
BAR 2 and 3	- memory decoder with size 1 MB - resource base 2^{20} … 2^{16}	
BAR 4 and 5	- IO decoder with size 64 K - resource base 2^{20} … 2^{16}	BAR 5 has switched off the I/O decoder.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXTDecoderDefaultSet
```

CLI Abbreviation

```
tdecdefset
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

See also

"BestXTDecoderGet" on page 95
"BestXTDecoderSet" on page 96
"BestXExerciserProg" on page 55
"BestXTDecoderAllSet" on page 93

BestXTDecoderGet

Call

```
bx_errtype BestXTDecoderGet(
      bx_handletype    handle,
      bx_dectype       dec,
      bx_decproptype   prop,
      bx_int32         *val );
```

Description Gets a property value of a target decoder from the host.

CLI Equivalent `BestXTDecoderGet dec=<dec> prop=<prop>`

CLI Abbreviation `tdecget dec=<dec> prop=<prop>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

dec Target decoder to be selected; see *“bx_dectype” on page 169*.

prop Target decoder property; see *“bx_decproptype” on page 168*.

Output Parameters

val Property value; see *“bx_decproptype” on page 168*.

See also

“BestXTDecoderDefaultSet” on page 94
“BestXTDecoderSet” on page 96
“BestXTDecoderAllSet” on page 93

BestXTDecoderSet

Call

```
bx_errtype BestXTDecoderSet(
      bx_handletype    handle,
      bx_dectype       dec,
      bx_decproptype   prop,
      bx_int32         val );
```

Description Sets a property of a target decoder on the host. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent `BestXTDecoderSet dec=<dec> prop=<prop> val=<val>`

CLI Abbreviation `tdecset dec=<dec> prop=<prop> val=<val>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

dec Target decoder to be selected; see *"bx_dectype" on page 169*.

prop Target decoder property; see *"bx_decproptype" on page 168*.

val Property value to be set; see *"bx_decproptype" on page 168*.

See also

"BestXTDecoderDefaultSet" on page 94
"BestXTDecoderGet" on page 95
"BestXExerciserProg" on page 55
"BestXTDecoderAllSet" on page 93

BestXCTSplitCondDefaultSet

Call

```
bx_errtype BestXCTSplitCondDefaultSet(
      bx_handletype   handle,
      bx_int32        dec );
```

Description

Sets the properties for one split decoder on the host to default values. For the default values, see *“bx_ctsplittype” on page 166*.

To write these settings to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXCTSplitCondDefaultSet dec=<dec>
```

CLI Abbreviation

```
ctsplitconddefset dec=<dec>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

dec Split decoder to be selected. Valid values are 0 … 3.

See also

“BestXCTSplitCondGet” on page 98
“BestXCTSplitCondSet” on page 99
“BestXExerciserProg” on page 55

BestXCTSplitCondGet

Call

```
bx_errtype BestXCTSplitCondGet(
      bx_handletype    handle,
      bx_int32         dec,
      bx_ctsplittype   prop,
      bx_int32         *val );
```

Description Gets a split decoder property for the selected split decoder from the host.

CLI Equivalent `BestXCTSplitCondGet dec=<dec> prop=<prop>`

CLI Abbreviation `ctsplitcondget dec=<dec> prop=<prop>`

Return Value Error code; see *"bx_errtype" on page 176.*

Input Parameters **handle** Session identification.

dec Split decoder to be selected. Valid values are 0 … 3. If several split decoders overlap in their decoding range, the decoder with the lower instance value has priority.

prop Split decoder property; see *"bx_ctsplittype" on page 166.*

Output Parameters **val** Property value; see *"bx_ctsplittype" on page 166.*

See also *"BestXCTSplitCondDefaultSet" on page 97*
"BestXCTSplitCondSet" on page 99

BestXCTSplitCondSet

Call

```
bx_errtype BestXCTSplitCondSet(
      bx_handletype    handle,
      bx_int32         dec,
      bx_ctsplittype   prop,
      bx_int32         val );
```

Description Sets a split decoder property for the selected split decoder on the host. To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent `BestXCTSplitCondSet dec=<dec> prop=<prop> val=<val>`

CLI Abbreviation `ctsplitcondset dec=<dec> prop=<prop> val=<val>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

dec Split decoder to be selected. Valid values are 0 … 3. If several split decoders overlap in their decoding range, the decoder with the lower instance value has priority.

prop Split decoder property; see *“bx_ctsplittype” on page 166*.

val Property value to be set; see *“bx_ctsplittype” on page 166*.

See also *“BestXCTSplitCondDefaultSet” on page 97*
“BestXCTSplitCondGet” on page 98
“BestXExerciserProg” on page 55

Expansion ROM Programming Functions

The following functions are used to program the expansion ROM flash memory space of the testcard:

Function	Action
"BestXExpRomWrite" on page 101	Writes a byte to the expansion ROM.
"BestXExpRomRead" on page 100	Reads a byte from the expansion ROM.

BestXExpRomRead

Call

```
bx_errtype BestXExpRomRead(
      bx_handletype   handle,
      bx_int32        offset,
      bx_int32        numbytes,
      bx_int8         *data );
```

Description Reads a specified number of bytes from the specified internal address in the expansion ROM flash memory.

CLI Equivalent `BestXExpRomRead offset=<offset> numbytes=<numbytes>`

CLI Abbreviation `expromread offset=<offset> numbytes=<numbytes>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

offset Offset in the expansion ROM memory. The offset needs to be WORD-aligned. Valid values are `0 … 65534`.

numbytes Number of bytes to read. Valid values are `0 … 65536`.

NOTE The sum of `offset` and `numbytes` must be ≤ `65536`.

Output Parameters **data** Return value.

Example `BestXExpRomRead offset=0 numbytes=8`

See also *"BestXExpRomWrite" on page 101*

BestXExpRomWrite

Call

```
bx_errtype BestXExpRomWrite(
      bx_handletype   handle,
      bx_int32        numbytes,
      bx_int8         *data );
```

Description Writes a specified number of bytes in the expansion ROM flash memory.

CLI Equivalent `BestXExpRomWrite numbytes=<numbytes> data=<[data_list]>`

CLI Abbreviation `expromwrite numbytes=<numbytes> data=<[data_list]>`

Return Value Error code; see *“bx_errtype” on page 176.*

Input Parameters

handle Session identification.

numbytes Number of bytes to read. Valid values are `0 … 65536`.

data Pointer to a 8-bit data array that contains the expansion ROM content.

Example

```
BestXExpRomWrite numbytes=8 data={0,1,2,3,4,5,6,7}
```

```
BestXExpRomWrite numbytes=8 data<"file.txt"
```

See also *“BestXExpRomRead” on page 100*

Data Memory Functions

The following functions access the data memory for initiator and target transactions:

Function	Action
"BestXDataMemInit" on page 102	Initializes the data memory of the testcard by filling it with zeros.
"BestXDataMemWrite" on page 104	Writes to the data memory of the testcard.
"BestXDataMemRead" on page 103	Reads from the data memory of the testcard.

BestXDataMemInit

Call `bx_errtype BestXDataMemInit( bx_handletype handle );`

Description Initializes the internal data memory of the testcard by filling it completely with zeros.

CLI Equivalent `BestXDataMemInit`

CLI Abbreviation `datameminit`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also *"BestXDataMemRead" on page 103*
"BestXDataMemWrite" on page 104

BestXDataMemRead

Call

```
bx_errtype BestXDataMemRead(
      bx_handletype   handle,
      bx_int32        addr,
      bx_int32        numbytes,
      bx_int8         *data );
```

Description Reads a data block from the data memory of the testcard to the host 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 `BestXDataMemRead addr=<addr> numbytes=<numbytes>`

CLI Abbreviation `datamemread addr=<addr> numbytes=<numbytes>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

addr Start address of source data within data memory of the testcard. The range depends on the available memory size.

numbytes Number of bytes to be read (maximum of `0x8000` when using the CLI).

Output Parameters **data** Destination buffer in the memory of the host. In C-API programs, you have to allocate a sufficient memory space.

See also *"BestXDataMemInit" on page 102*
"BestXDataMemWrite" on page 104

BestXDataMemWrite

Call

```
bx_errtype BestXDataMemWrite(
      bx_handletype   handle,
      bx_int32        addr,
      bx_int32        numbytes,
      bx_int8         *data );
```

Description

Writes a data block from the memory on the host to the data memory of the testcard. The data memory is shared by initiator and target. Both can use the data for further testing.

CLI Equivalent

```
BestXDataMemWrite addr=<addr> numbytes=<numbytes>
data=<{data_list}>
```

CLI Abbreviation

```
datamemwrite addr=<addr> numbytes=<numbytes> data=<{data_list}>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

addr Destination start address for the data within the data memory of the testcard. The range depends on the available memory size.

numbytes Number of bytes to be written.

data Source data in the system memory of the host. You have to allocate appropriate memory space.

data_ptr **CLI only.** List of data to be transferred when using the CLI (for example, `data={0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8}`).

See also

"BestXDataMemInit" on page 102
"BestXDataMemRead" on page 103

Host Access Functions

The following functions are used for data transfer to and from the host (control PC):

Function	Action
“BestXHostPCIRegWrite” on page 107	Writes a register of a PCI-X device.
“BestXHostPCIRegRead” on page 106	Reads a register of a PCI-X device.

BestXHostPCIRegRead

Call

```
bx_errtype BestXHostPCIRegRead(
      bx_handletype      handle,
      bx_addrspacetype   addrspace,
      bx_int32           addr,
      bx_int32           *val,
      bx_sizetype        size );
```

Description

Reads the value from a specific PCI-X device register in a 32-bit address space on the host—the type of address space determines a Configuration, Memory or I/O Read.

The bus address is a byte address. This function performs single-cycle transactions and sets automatically the correct byte enables corresponding to the word size and bus address.

This function only applies to a 32-bit address space.

CLI Equivalent

```
BestXHostPCIRegRead addrspace=<addrspace> addr=<addr> size=<size>
```

CLI Abbreviation

```
hostpciregread addrspace=<addrspace> addr=<addr> size=<size>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

space Type of address space for the register access; see *“bx_addrspacetype” on page 159*.

addr PCI-X bus or system address.

size Defines the width of the value to be obtained from the register; see *“bx_sizetype” on page 195*.

Output Parameters

val Value obtained from the register.

See also

“BestXHostPCIRegWrite” on page 107

BestXHostPCIRegWrite

Call

```
bx_errtype BestXHostPCIRegWrite(
      bx_handletype      handle,
      bx_addrspacetype   addrspace,
      bx_int32           addr,
      bx_int32           val,
      bx_sizetype        size );
```

Description

Writes a value to a specific PCI-X device register on the host—the type of address space determines a Configuration Write, Memory Write or I/O Write.

The bus address is a byte address. This function performs single-cycle transactions and sets automatically the correct byte enables corresponding to word size and bus address.

This function only applies to a 32-bit address space.

To write the property to the testcard, use the `BestXExerciserProg` call.

CLI Equivalent

```
BestXHostPCIRegWrite addrspace=<addrspace> addr=<addr> val=<val>
size=<size>
```

CLI Abbreviation

```
hostpciregwrite addrspace=<addrspace> addr=<addr> val=<val>
size=<size>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

space Type of address space for the register access; see *“bx_addrspacetype” on page 159*.

addr PCI-X bus or system address.

val Value to be set to the register.

size Defines the width of the values to be set in the register; see *“bx_sizetype” on page 195*.

See also

“BestXHostPCIRegRead” on page 106
“BestXExerciserProg” on page 55

Protocol Permutator and Randomizer Functions

The PCI-X protocol permutator and randomizer (PPR) functions are divided as follows:

PPR Functions	Action
PPR Administration Functions	Initialize and deinitialize the PPR software.
PPR Generic Functions	Prepare for PPR programming.
PPR Requester-Initiator Functions	Prepare and perform requester-initiator block and behavior permutations.
PPR Requester-Target Functions	Prepare and perform requester-target behavior permutations.
PPR Completer-Initiator Functions	Prepare and perform completer-initiator behavior permutations.
PPR Completer-Target Functions	Prepare and perform completer-target behavior permutations.
PPR Reporting Functions	Generate reports.

PPR Administration Functions

The following functions are used to initialize and deinitialize the PCI-X Protocol Permutator and Randomizer software:

Function	Action
"BestXPprInit" on page 111	Initializes the PPR software.
"BestXPprProg" on page 111	Writes all current PPR settings to the testcard.
"BestXPprDelete" on page 110	Frees all memory allocated by the software.

BestXPprDelete

Call `bx_errtype BestXPprDelete( bx_handletype handle );`

Description Frees all the memory space allocated by the PCI-X PPR software for the current testcard. Use this function when you finish working with the PPR software.

CLI Equivalent `BestXPprDelete`

CLI Abbreviation `pprdelete`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also *"BestXPprInit" on page 111*

BestXPprInit

Call `bx_errtype BestXPprInit( bx_handletype handle );`

Description Initializes the PCI-X PPR software for the connected testcard and sets all PPR software properties to default values. This function must be called before any other PCI-X PPR function.

At the end of the test, you must call `BestXPprDelete` to free the memory.

CLI Equivalent `BestXPprInit`

CLI Abbreviation `pprinit`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also *"BestXPprDelete" on page 110*

BestXPprProg

Call `bx_errtype BestXPprProg( bx_handletype handle );`

Description Writes the current PPR settings to the exerciser testcard. To use this function, the exerciser must be stopped (see *"BestXExerciserStop" on page 57*).

You can define which parts of the exerciser are programmed by setting the `BXPPR_GEN_USE_xx` generic properties (see *"BestXPprGenSet" on page 114*).

CLI Equivalent `BestXPprProg`

CLI Abbreviation `pprprog`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also -

PPR Generic Functions

The following functions are used to prepare for PPR programming:

Function	Action
"BestXPprGenDefaultSet" on page 112	Sets all generic PPR properties to default values.
"BestXPprGenSet" on page 114	Sets a generic PPR property.
"BestXPprGenGet" on page 113	Gets a generic PPR property.

BestXPprGenDefaultSet

Call

```
bx_errtype BestXPprGenDefaultSet( bx_handletype handle );
```

Description

Sets all generic properties of the PCI-X PPR software to default values. For the default values, see *"bxppr_gentype" on page 203*.

CLI Equivalent

```
BestXPprGenDefaultSet
```

CLI Abbreviation

```
pprgendefset
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

See also

"BestXPprGenSet" on page 114
"BestXPprGenGet" on page 113
"BestXPprProg" on page 111

BestXPprGenGet

Call

```
bx_errtype BestXPprGenGet(
      bx_handletype   handle,
      bxppr_gentype   genprop,
      bx_int32        *pValue );
```

Description

Gets generic property values of the PCI-X PPR software.

CLI Equivalent

```
BestXPprGenGet prop=<genprop>
```

CLI Abbreviation

```
pprgenget prop=<genprop>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

genprop Property to be obtained; see *"bxppr_gentype" on page 203*.

Output Parameters

pValue Value of the property; see *"bxppr_gentype" on page 203*.

See also

"BestXPprGenSet" on page 114
"BestXPprGenDefaultSet" on page 112

BestXPprGenSet

Call

```
bx_errtype BestXPprGenSet(
      bx_handletype   handle,
      bxppr_gentype   genprop,
      bx_int32        value );
```

Description Sets generic property values of the PCI-X PPR software.

CLI Equivalent `BestXPprGenSet prop=<genprop> value=<value>`

CLI Abbreviation `pprgenset prop=<genprop> value=<value>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Handle to identify the session.

prop Property to be set; see *“bxppr_gentype” on page 203*.

value Value of the property to be set; see *“bxppr_gentype” on page 203*.

See also

“BestXPprGenDefaultSet” on page 112
“BestXPprGenGet” on page 113
“BestXPprProg” on page 111

PPR Requester-Initiator Functions

The following functions are used to prepare and to perform requester-initiator block and behavior permutations:

Function	Action
“BestXPprRIDefaultSet” on page 127	Sets all PPR requester-initiator properties to default values.
“BestXPprRIBlkPermDefaultSet” on page 125	Sets all requester-initiator block permutation properties to default values.
“BestXPprRIBlkPermSet” on page 126	Sets a requester-initiator block permutation property.
“BestXPprRIBlkPermGet” on page 125	Gets a requester-initiator block permutation property.
“BestXPprRIBlkListDefaultSet” on page 122	Sets all list values from a requester-initiator block property to default values.
“BestXPprRIBlkListSet” on page 124	Sets the list values from a requester-initiator block property.
“BestXPprRIBlkListGet” on page 123	Gets the list values from a requester-initiator block property.
“BestXPprRIBlkResultGet” on page 127	Gets the permutation result from a requester-initiator result property.
“BestXPprRIBlkGapGet” on page 128	Gets a gap property for a specific gap.
“BestXPprRIBehPermDefaultSet” on page 119	Sets all requester-initiator behavior permutation properties to default values.
“BestXPprRIBehPermSet” on page 120	Sets a requester-initiator behavior permutation property.
“BestXPprRIBehPermGet” on page 119	Gets a requester-initiator behavior permutation property.
“BestXPprRIBehListDefaultSet” on page 116	Sets the value list of a requester-initiator behavior property to default values.
“BestXPprRIBehListSet” on page 118	Sets the value list of a requester-initiator behavior property.
“BestXPprRIBehListGet” on page 117	Gets the value list of a requester-initiator behavior property.
“BestXPprRIBehResultGet” on page 121	Gets the permutation result of a PPR requester-initiator behavior result property.

BestXPprRIBehListDefaultSet

Call

```
bx_errtype BestXPprRIBehListDefaultSet(
      bx_handletype   handle,
      bx_ribehtype    behavior );
```

Description

Sets all variation list values for a PPR requester-initiator behavior property on the host to default values. For default values, see *"bx_ribehtype" on page 182*.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprRIBehListDefaultSet behavior=<behavior>
```

CLI Abbreviation

```
pprribehlistdefaultset beh=<behavior>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

behavior Behavior property to be set; see *"bx_ribehtype" on page 182*.

See also

"BestXPprRIBehListSet" on page 118
"BestXPprRIBehListGet" on page 117
"BestXPprProg" on page 111

BestXPprRIBehListGet

Call

```
bx_errtype BestXPprRIBehListGet(
      bx_handletype    handle,
      bx_ribehtype     behavior,
      bxppr_listtype   *pList );
```

Description Gets the variation list for a PPR requester-initiator behavior property from the host.

CLI Equivalent `BestXPprRIBehListGet behavior=<behavior>`

CLI Abbreviation `pprribehlistget beh=<behavior>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

behavior Behavior property to be obtained; see *"bx_ribehtype" on page 182*.

Output Parameters **pList** Value list of the behavior property. For the syntax of the list, see *"bxppr_listtype" on page 204*.

See also *"BestXPprRIBehListDefaultSet" on page 116*
"BestXPprRIBehListSet" on page 118

BestXPprRIBehListSet

Call

```
bx_errtype BestXPprRIBehListSet(
      bx_handletype    handle,
      bx_ribehtype     behavior,
      bxppr_listtype   *pList );
```

Description

Sets the variation list for a PPR requester-initiator behavior property on the host.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprRIBehListSet behavior=<behavior> list=<list>
```

CLI Abbreviation

```
pprribehlistset beh=<behavior> list=<list>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

behavior Behavior property to be set; see *“bx_ribehtype” on page 182*.

pList Value list of the behavior property to be set. For the syntax of the list, see *“bxppr_listtype” on page 204*.

See also

“BestXPprRIBehListDefaultSet” on page 116
“BestXPprRIBehListGet” on page 117
“BestXPprProg” on page 111

BestXPprRIBehPermDefaultSet

Call

```
bx_errtype BestXPprRIBehPermDefaultSet( bx_handletype handle );
```

Description

Sets all PPR requester-initiator behavior properties on the host to default values. For default values, see *“bxppr_behpermtype” on page 201*.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprRIBehPermDefaultSet
```

CLI Abbreviation

```
pprribehpermdefaultset
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

See also

“BestXPprRIBehPermSet” on page 120
“BestXPprRIBehPermGet” on page 119
“BestXPprProg” on page 111

BestXPprRIBehPermGet

Call

```
bx_errtype BestXPprRIBehPermGet(
      bx_handletype       handle,
      bxppr_behpermtype   permprop,
      bx_int32            *pValue );
```

Description

Gets a PPR requester-initiator behavior property from the host.

CLI Equivalent

```
BestXPprRIBehPermGet prop=<permprop>
```

CLI Abbreviation

```
pprribehpermget prop=<permprop>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

permprop Behavior property to be obtained; see *“bxppr_behpermtype” on page 201*.

Output Parameters

pValue Value of the behavior property; see *“bxppr_behpermtype” on page 201*.

See also

“BestXPprRIBehPermDefaultSet” on page 119
“BestXPprRIBehPermSet” on page 120

BestXPprRIBehPermSet

Call

```
bx_errtype BestXPprRIBehPermSet(
      bx_handletype       handle,
      bxppr_behpermtype   permprop,
      bx_int32            value );
```

Description

Sets a PPR requester-initiator behavior property on the host.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprRIBehPermSet prop=<permprop> value=<value>
```

CLI Abbreviation

```
pprribehpermset prop=<permprop> value=<value>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

permprop Behavior property to be set; see *"bxppr_behpermtype" on page 201*.

value Value of the behavior property to be set; see *"bxppr_behpermtype" on page 201*.

See also

"BestXPprRIBehPermDefaultSet" on page 119
"BestXPprRIBehPermGet" on page 119
"BestXPprProg" on page 111

BestXPprRIBehResultGet

Call

```
bx_errtype BestXPprRIBehResultGet(
      bx_handletype         handle,
      bxppr_behresulttype   resultprop,
      bx_int32              *pValue );
```

Description Gets the permutation result for the specific PPR requester-initiator behavior result property from the host.

CLI Equivalent `BestXPprRIBehResultGet prop=<resultprop>`

CLI Abbreviation `pprribehresultget prop=<resultprop>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

resultprop Behavior result property to be obtained; see *"bxppr_behresulttype" on page 202*.

Output Parameters **pValue** Value of the behavior result property; *"bxppr_behresulttype" on page 202*.

See also *"BestXPprRIBlkResultGet" on page 127*

BestXPprRIBlkListDefaultSet

Call

```
bx_errtype BestXPprRIBlkListDefaultSet(
      bx_handletype     handle,
      bxppr_riblktype   blkprop );
```

Description Sets the variation lists of all PPR block requester-initiator properties on the host to default values.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent `BestXPprRIBlkListDefaultSet prop=<blkprop>`

CLI Abbreviation `pprriblklistdefaultset prop=<blkprop>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

blkprop Block property from which the list values are to be set; see *“bxppr_riblktype” on page 210*.

See also *“BestXPprRIBlkListSet” on page 124*
“BestXPprRIBlkListGet” on page 123
“BestXPprProg” on page 111

BestXPprRIBlkListGet

Call

```
bx_errtype BestXPprRIBlkListGet(
      bx_handletype     handle,
      bxppr_riblktype   blkprop,
      bxppr_listtype    *pList );
```

Description Gets a variation list of a PPR block requester-initiator property from the host.

CLI Equivalent `BestXPprRIBlkListGet prop=<blkprop>`

CLI Abbreviation `pprriblklistget prop=<blkprop>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

blkprop Property from which the list values are to be obtained; see *"bxppr_riblktype" on page 210*.

Output Parameters **pList** List values of the property. For the syntax of the list, see *"bxppr_listtype" on page 204*.

See also *"BestXPprRIBlkListDefaultSet" on page 122*
"BestXPprRIBlkListSet" on page 124

BestXPprRIBlkListSet

Call

```
bx_errtype BestXPprRIBlkListSet(
      bx_handletype     handle,
      bxppr_riblktype   blkprop,
      bxppr_listtype    *pList );
```

Description

Sets the variation list of a PPR block requester-initiator property on the host.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprRIBlkListSet prop=<blkprop> list=<list>
```

CLI Abbreviation

```
pprriblklistset prop=<blkprop> list=<list>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

blkprop Block property from which the list values are to be set; see *“bxppr_riblktype” on page 210*.

pList List values of the property to be set. For the syntax of the list, see *“bxppr_listtype” on page 204*.

See also

“BestXPprRIBlkListDefaultSet” on page 122
“BestXPprRIBlkListGet” on page 123
“BestXPprProg” on page 111

BestXPprRIBlkPermDefaultSet

Call

```
bx_errtype BestXPprRIBlkPermDefaultSet( bx_handletype handle );
```

Description

Sets all PPR requester-initiator block permutation properties on the host to default values. For the default values, see *"bxppr_riblkpermtype" on page 208*.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprRIBlkPermDefaultSet
```

CLI Abbreviation

```
pprriblkpermdefaultset
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

See also

"BestXPprRIBlkPermSet" on page 126
"BestXPprRIBlkPermGet" on page 125
"BestXPprProg" on page 111

BestXPprRIBlkPermGet

Call

```
bx_errtype BestXPprRIBlkPermGet(
      bx_handletype         handle,
      bxppr_riblkpermtype   permprop,
      bx_int32              *pValue );
```

Description

Gets a PPR requester-initiator block permutation property from the host.

CLI Equivalent

```
BestXPprRIBlkPermGet prop=<permprop>
```

CLI Abbreviation

```
pprriblkpermget prop=<permprop>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

permprop Block permutation property to be obtained; see *"bxppr_riblkpermtype" on page 208*.

Output Parameters

pValue Value of the property; see *"bxppr_riblkpermtype" on page 208*.

See also

"BestXPprRIBlkPermDefaultSet" on page 125
"BestXPprRIBlkPermSet" on page 126

BestXPprRIBlkPermSet

Call

```
bx_errtype BestXPprRIBlkPermSet(
      bx_handletype         handle,
      bxppr_riblkpermtype   permprop,
      bx_int32              value );
```

Description

Sets a PPR requester-initiator block property on the host.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprRIBlkPermSet prop=<permprop> value=<value>
```

CLI Abbreviation

```
pprriblkpermset prop=<permprop> value=<value>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

permprop Block permutation property to be set; see *“bxppr_riblkpermtype” on page 208*.

value Value of the property to be set; see *“bxppr_riblkpermtype” on page 208*.

See also

“BestXPprRIBlkPermDefaultSet” on page 125
“BestXPprRIBlkPermGet” on page 125
“BestXPprProg” on page 111

BestXPprRIBlkResultGet

Call

```
bx_errtype BestXPprRIBlkResultGet(
      bx_handletype           handle,
      bxppr_riblkresulttype   resultprop,
      bx_int32                *pValue );
```

Description Gets the permutation result for the specific block result property from the host.

CLI Equivalent `BestXPprRIBlkResultGet prop=<resultprop>`

CLI Abbreviation `pprriblkresultget prop=<resultprop>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

resultprop Block result property from which the results are to be obtained; see *“bxppr_riblkresulttype” on page 209*.

Output Parameters **pValue** Result of the property; see *“bxppr_riblkresulttype” on page 209*.

See also –

BestXPprRIDefaultSet

Call

```
bx_errtype BestXPprRIDefaultSet( bx_handletype handle );
```

Description Sets all PPR requester-initiator properties on the host to default values.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent `BestXPprRIDefaultSet`

CLI Abbreviation `pprridefaultset`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

See also *“BestXPprRIBlkPermDefaultSet” on page 125*
“BestXPprRIDefaultSet” on page 127
“BestXPprRIBehPermDefaultSet” on page 119
“BestXPprRIBehListDefaultSet” on page 116
“BestXPprProg” on page 111

BestXPprRIBlkGapGet

Call

```
bx_errtype BestXPprRIBlkGapGet(
      bx_handletype        handle,
      bx_int32             gapnum,
      bxppr_riblkgaptype   gapprop,
      bx_int32             *pValue );
```

Description Gets the gap property for the specified gap from the host.

CLI Equivalent `BestXPprRIBlkGapGet gapnum=<gapnum> type=<gapprop>`

CLI Abbreviation `pprriblkgapget gapnum=<gapnum> type=<gapprop>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

gapnum Number of the gap that is referenced.

gapprop Gap property to be obtained; see *"bxppr_riblkgaptype" on page 206*.

Output Parameters

pValue Value of the gap property; see *"bxppr_riblkgaptype" on page 206*.

See also –

PPR Requester-Target Functions

The following functions are used to prepare and to perform requester-target behavior permutations:

Function	Action
"BestXPprRTDefaultSet" on page 136	Sets all PPR requester-target properties to default values.
"BestXPprRTBehPermDefaultSet" on page 133	Sets all PPR requester-target behavior properties to default values.
"BestXPprRTBehPermSet" on page 135	Sets a PPR requester-target behavior property.
"BestXPprRTBehPermGet" on page 134	Gets a PPR requester-target behavior property.
"BestXPprRTBehListDefaultSet" on page 130	Sets all list values of PPR requester-target behavior properties to default values.
"BestXPprRTBehListSet" on page 132	Sets the list values of a PPR requester-target behavior property.
"BestXPprRTBehListGet" on page 131	Gets the list values of a PPR requester-target behavior property.
"BestXPprRTBehResultGet" on page 136	Gets the permutation result of a PPR requester-target behavior result property.

BestXPprRTBehListDefaultSet

Call

```
bx_errtype BestXPprRIBehListDefaultSet(
      bx_handletype   handle,
      bx_rtbehtype    behavior );
```

Description

Sets all variation list values for a PPR requester-target behavior property on the host to default values. For default values, see *“bx_rtbehtype” on page 188*.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprRTBehListDefaultSet behavior=<behavior>
```

CLI Abbreviation

```
pprrtbehlistdefaultset beh=<behavior>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

behavior Behavior property to be set; see *“bx_rtbehtype” on page 188*.

See also

“BestXPprRTBehListSet” on page 132
“BestXPprRTBehListGet” on page 131
“BestXPprProg” on page 111

BestXPprRTBehListGet

Call

```
bx_errtype BestXPprRTBehListGet(
      bx_handletype    handle,
      bx_rtbehtype     behavior,
      bxppr_listtype   *pList );
```

Description

Gets the variation list for a PPR requester-target behavior property from the host.

CLI Equivalent

```
BestXPprRTBehListGet behavior=<behavior>
```

CLI Abbreviation

```
pprrtbehlistget beh=<behavior>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

behavior Behavior property to be obtained; see *“bx_rtbehtype” on page 188*.

Output Parameters

pList Value list of the behavior property. For the syntax of the list, see *“bxppr_listtype” on page 204*.

See also

“BestXPprRTBehListDefaultSet” on page 130
“BestXPprRTBehListSet” on page 132

BestXPprRTBehListSet

Call

```
bx_errtype BestXPprRTBehListSet(
      bx_handletype    handle,
      bx_rtbehtype     behavior,
      bxppr_listtype   *pList );
```

Description

Sets the variation list for a PPR requester-target behavior property on the host.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprRTBehListSet behavior=<behavior> list=<list>
```

CLI Abbreviation

```
pprrtbehlistset beh=<behavior> list=<list>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

behavior Behavior property to be set; see *"bx_rtbehtype" on page 188*.

pList Value list of the behavior property to be set. For the syntax of the list, see *"bxppr_listtype" on page 204*.

See also

"BestXPprRTBehListDefaultSet" on page 130
"BestXPprRTBehListGet" on page 131
"BestXPprProg" on page 111

BestXPprRTBehPermDefaultSet

Call

```
bx_errtype BestXPprRTBehPermDefaultSet(
      bx_handletype       handle,
      bxppr_behpermtype   permprop,
      bx_int32            value );
```

Description

Sets all PPR requester-initiator behavior permutation properties on the host to default values. For default values, see *"bxppr_behpermtype" on page 201*.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

`BestXPprRIBehPermDefaultSet`

CLI Abbreviation

`pprribehpermdefaultset`

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

See also

"BestXPprRTBehPermSet" on page 135
"BestXPprRTBehPermGet" on page 134
"BestXPprProg" on page 111

BestXPprRTBehPermGet

Call

```
bx_errtype BestXPprRTBehPermGet(
      bx_handletype       handle,
      bxppr_behpermtype   permprop,
      bx_int32            *pValue );
```

Description Gets a PPR requester-target behavior permutation property from the host.

CLI Equivalent `BestXPprRTBehPermGet prop=<permprop>`

CLI Abbreviation `pprrtbehpermget prop=<permprop>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

permprop Behavior permutation property to be obtained; see *“bxppr_behpermtype” on page 201*.

Output Parameters

pValue Value of the behavior permutation property; see *“bxppr_behpermtype” on page 201*.

See also

“BestXPprRTBehPermDefaultSet” on page 133
“BestXPprRTBehPermSet” on page 135

BestXPprRTBehPermSet

Call

```
bx_errtype BestXPprRTBehPermSet(
      bx_handletype       handle,
      bxppr_behpermtype   permprop,
      bx_int32            value );
```

Description Sets a PPR requester-target behavior permutation property on the host.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent `BestXPprRTBehPermSet prop=<permprop> value=<value>`

CLI Abbreviation `pprrtbehpermset prop=<permprop> value=<value>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

permprop Behavior permutation property to be set; see *“bxppr_behpermtype” on page 201*.

value Value of the behavior permutation property to be set; see *“bxppr_behpermtype” on page 201*.

See also *“BestXPprRTBehPermDefaultSet” on page 133*
“BestXPprRTBehPermGet” on page 134
“BestXPprProg” on page 111

BestXPprRTBehResultGet

Call

```
bx_errtype BestXPprRTBehResultGet(
      bx_handletype           handle,
      bxppr_behresulttype   resultprop,
      bx_int32                *pValue );
```

Description Gets the permutation result for the specific PPR requester-target behavior result property from the host.

CLI Equivalent `BestXPprRTBehResultGet prop=<resultprop>`

CLI Abbreviation `pprrtbehresultget prop=<resultprop>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

resultprop Behavior result property to be obtained; see *“bxppr_behresulttype” on page 202*.

Output Parameters **pValue** Value of the behavior result property; *“bxppr_behresulttype” on page 202*.

See also *“BestXPprRIBlkResultGet” on page 127*

BestXPprRTDefaultSet

Call

```
bx_errtype BestXPprRTDefaultSet( bx_handletype handle );
```

Description Sets all PPR requester-target properties on the host to default values.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent `BestXPprRTDefaultSet`

CLI Abbreviation `pprrtdefaultset`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

See also *“BestXPprRTBehPermDefaultSet” on page 133*
“BestXPprRIBehPermDefaultSet” on page 119
“BestXPprRTBehListDefaultSet” on page 130
“BestXPprProg” on page 111

PPR Completer-Target Functions

The following functions are used to prepare and to perform completer-target behavior permutations:

Function	Action
“BestXPprCTDefaultSet” on page 143	Sets all PPR completer-target properties to default values.
“BestXPprCTBehPermDefaultSet” on page 141	Sets all completer-target behavior permutation properties to default values.
“BestXPprCTBehPermSet” on page 142	Sets a completer-target behavior permutation property.
“BestXPprCTBehPermGet” on page 141	Gets a completer-target behavior permutation property.
“BestXPprCTBehListDefaultSet” on page 138	Sets the value list of a completer-target behavior property to default values.
“BestXPprCTBehListSet” on page 140	Sets the value list of a completer-target behavior property.
“BestXPprCTBehListGet” on page 139	Gets the value list of a completer-target behavior property.
“BestXPprCTBehResultGet” on page 143	Gets the permutation result of a PPR completer-target behavior result property.

BestXPprCTBehListDefaultSet

Call

```
bx_errtype BestXPprCTBehListDefaultSet(
      bx_handletype   handle,
      bx_ctbehtype    behavior );
```

Description

Sets all variation list values for a PPR completer-target behavior property on the host to default values. For default values, see *“bx_ctbehtype” on page 163*.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprCTBehListDefaultSet behavior=<behavior>
```

CLI Abbreviation

```
pprcibehlistdefaultset beh=<behavior>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

behavior Behavior property to be set; see *“bx_ctbehtype” on page 163*.

See also

“BestXPprCTBehListSet” on page 140
“BestXPprCTBehListGet” on page 139
“BestXPprProg” on page 111

BestXPprCTBehListGet

Call

```
bx_errtype BestXPprCTBehListGet(
      bx_handletype    handle,
      bx_ctbehtype     behavior,
      bxppr_listtype   *pList );
```

Description Gets the variation list for a PPR completer-target behavior property from the host.

CLI Equivalent `BestXPprCTBehListGet behavior=<behavior>`

CLI Abbreviation `pprctbehlistget beh=<behavior>`

Return Value Error code; see *"bx_errtype" on page 176.*

Input Parameters **handle** Session identification.

behavior Behavior property to be obtained; see *"bx_ctbehtype" on page 163.*

Output Parameters **pList** Value list of the behavior property. For the syntax of the list, see *"bxppr_listtype" on page 204.*

See also *"BestXPprCTBehListDefaultSet" on page 138*
"BestXPprCTBehListSet" on page 140

BestXPprCTBehListSet

Call

```
bx_errtype BestXPprCTBehListSet(
      bx_handletype    handle,
      bx_ctbehtype     behavior,
      bxppr_listtype   *pList );
```

Description

Sets the variation list for a PPR completer-target behavior property on the host.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprCTBehListSet behavior=<behavior> list=<list>
```

CLI Abbreviation

```
pprctbehlistset beh=<behavior> list=<list>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

behavior Behavior property to be set; see *“bx_ctbehtype” on page 163*.

pList Value list of the behavior property to be set. For the syntax of the list, see *“bxppr_listtype” on page 204*.

See also

“BestXPprCTBehListDefaultSet” on page 138
“BestXPprCTBehListGet” on page 139
“BestXPprProg” on page 111

BestXPprCTBehPermDefaultSet

Call

```
bx_errtype BestXPprCTBehPermDefaultSet( bx_handletype handle );
```

Description

Sets all PPR completer-target behavior permutation properties on the host to default values. For default values, see *"bxppr_behpermtype" on page 201*.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprCTBehPermDefaultSet
```

CLI Abbreviation

```
pprctbehpermdefaultset
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

See also

"BestXPprCTBehPermSet" on page 142
"BestXPprCTBehPermGet" on page 141
"BestXPprProg" on page 111

BestXPprCTBehPermGet

Call

```
bx_errtype BestXPprCTBehPermGet(
      bx_handletype       handle,
      bxppr_behpermtype   permprop,
      bx_int32            *pValue );
```

Description

Gets a PPR completer-target behavior permutation property from the host.

CLI Equivalent

```
BestXPprCTBehPermGet prop=<permprop>
```

CLI Abbreviation

```
pprctbehpermget prop=<permprop>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

permprop Behavior property to be obtained; see *"bxppr_behpermtype" on page 201*.

Output Parameters

pValue Value of the behavior permutation property; see *"bxppr_behpermtype" on page 201*.

See also

"BestXPprCTBehPermDefaultSet" on page 141
"BestXPprCTBehPermSet" on page 142

BestXPprCTBehPermSet

Call

```
bx_errtype BestXPprCTBehPermSet(
      bx_handletype       handle,
      bxppr_behpermtype   permprop,
      bx_int32            value );
```

Description

Sets a PPR completer-target behavior property on the host.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprCTBehPermSet prop=<permprop> value=<value>
```

CLI Abbreviation

```
pprctbehpermset prop=<permprop> value=<value>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

permprop Behavior property to be set; see *“bxppr_behpermtype” on page 201*.

value Value of the behavior property to be set; see *“bxppr_behpermtype” on page 201*.

See also

“BestXPprCTBehPermDefaultSet” on page 141
“BestXPprCTBehPermGet” on page 141
“BestXPprProg” on page 111

BestXPprCTBehResultGet

Call

```
bx_errtype BestXPprCTBehResultGet(
      bx_handletype         handle,
      bxppr_behresulttype   resultprop,
      bx_int32              *pValue );
```

Description Gets the permutation result for the specific PPR completer-target behavior result property from the host.

CLI Equivalent `BestXPprCTBehResultGet prop=<resultprop>`

CLI Abbreviation `pprctbehresultget prop=<resultprop>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

resultprop Behavior result property to be obtained; see *"bxppr_behresulttype" on page 202*.

Output Parameters **pValue** Value of the behavior result property; *"bxppr_behresulttype" on page 202*.

See also –

BestXPprCTDefaultSet

Call

```
bx_errtype BestXPprCTDefaultSet( bx_handletype handle );
```

Description Sets all PPR completer-target properties on the host to default values.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent `BestXPprCTDefaultSet`

CLI Abbreviation `pprctdefaultset`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also *"BestXPprCTBehPermDefaultSet" on page 141*
"BestXPprCTBehListDefaultSet" on page 138
"BestXPprProg" on page 111

PPR Completer-Initiator Functions

The following functions are used to prepare and to perform completer-initiator behavior permutations:

Function	Action
"BestXPprCIDefaultSet" on page 150	Sets all PPR completer-initiator properties to default values.
"BestXPprCIBehPermDefaultSet" on page 148	Sets all completer-initiator behavior permutation properties to default values.
"BestXPprCIBehPermSet" on page 149	Sets a completer-initiator behavior permutation property.
"BestXPprCIBehPermGet" on page 148	Gets a completer-initiator behavior permutation property.
"BestXPprCIBehListDefaultSet" on page 145	Sets the value list of a completer-initiator behavior property to default values.
"BestXPprCIBehListSet" on page 147	Sets the value list of a completer-initiator behavior property.
"BestXPprCIBehListGet" on page 146	Gets the value list of a completer-initiator behavior property.
"BestXPprCIBehResultGet" on page 150	Gets the permutation result of a PPR completer-initiator behavior result property.

BestXPprCIBehListDefaultSet

Call

```
bx_errtype BestXPprCIBehListDefaultSet(
      bx_handletype   handle,
      bx_cibehtype    behavior );
```

Description

Sets all variation list values for a PPR completer-initiator behavior property on the host to default values. For default values, see *“bx_cibehtype” on page 161.*

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprCIBehListDefaultSet behavior=<behavior>
```

CLI Abbreviation

```
pprcibehlistdefaultset beh=<behavior>
```

Return Value

Error code; see *“bx_errtype” on page 176.*

Input Parameters

handle Session identification.

behavior Behavior property to be set; see *“bx_cibehtype” on page 161.*

See also

“BestXPprCIBehListSet” on page 147
“BestXPprCIBehListGet” on page 146
“BestXPprProg” on page 111

BestXPprCIBehListGet

Call

```
bx_errtype BestXPprCIBehListGet(
      bx_handletype    handle,
      bx_cibehtype     behavior,
      bxppr_listtype   *pList );
```

Description Gets the variation list for a PPR completer-initiator behavior property from the host.

CLI Equivalent `BestXPprCIBehListGet behavior=<behavior>`

CLI Abbreviation `pprcibehlistget beh=<behavior>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

behavior Behavior property to be obtained; see *“bx_cibehtype” on page 161*.

Output Parameters **pList** Value list of the behavior property. For the syntax of the list, see *“bxppr_listtype” on page 204*.

See also *“BestXPprCIBehListDefaultSet” on page 145*
“BestXPprCIBehListSet” on page 147

BestXPprCIBehListSet

Call

```
bx_errtype BestXPprCIBehListSet(
      bx_handletype    handle,
      bx_cibehtype     behavior,
      bxppr_listtype   *pList );
```

Description

Sets the variation list for a PPR completer-initiator behavior property on the host.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprCIBehListSet behavior=<behavior> list=<list>
```

CLI Abbreviation

```
pprcibehlistset beh=<behavior> list=<list>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

behavior Behavior property to be set; see *"bx_cibehtype" on page 161*.

pList Value list of the behavior property to be set. For the syntax of the list, see *"bxppr_listtype" on page 204*.

See also

"BestXPprCIBehListDefaultSet" on page 145
"BestXPprCIBehListGet" on page 146
"BestXPprProg" on page 111

BestXPprCIBehPermDefaultSet

Call

```
bx_errtype BestXPprCIBehPermDefaultSet( bx_handletype handle );
```

Description

Sets all PPR completer-initiator behavior properties on the host to default values. For default values, see *"bxppr_behpermtype" on page 201*.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprCIBehPermDefaultSet
```

CLI Abbreviation

```
pprcibehpermdefaultset
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

See also

"BestXPprCIBehPermSet" on page 149
"BestXPprCIBehPermGet" on page 148
"BestXPprProg" on page 111

BestXPprCIBehPermGet

Call

```
bx_errtype BestXPprCIBehPermGet(
      bx_handletype       handle,
      bxppr_behpermtype   permprop,
      bx_int32            *pValue );
```

Description

Gets a PPR completer-initiator behavior property from the host.

CLI Equivalent

```
BestXPprCIBehPermGet prop=<permprop>
```

CLI Abbreviation

```
pprcibehpermget prop=<permprop>
```

Return Value

Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

permprop Behavior property to be obtained; see *"bxppr_behpermtype" on page 201*.

Output Parameters

pValue Value of the behavior property; see *"bxppr_behpermtype" on page 201*.

See also

"BestXPprCIBehPermDefaultSet" on page 148
"BestXPprCIBehPermSet" on page 149

BestXPprCIBehPermSet

Call

```
bx_errtype BestXPprCIBehPermSet(
      bx_handletype       handle,
      bxppr_behpermtype   permprop,
      bx_int32            value );
```

Description

Sets a PPR completer-initiator behavior property on the host.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent

```
BestXPprCIBehPermSet prop=<permprop> value=<value>
```

CLI Abbreviation

```
pprcibehpermset prop=<permprop> value=<value>
```

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

permprop Behavior property to be set; see *“bxppr_behpermtype” on page 201*.

value Value of the behavior property to be set; see *“bxppr_behpermtype” on page 201*.

See also

“BestXPprCIBehPermDefaultSet” on page 148
“BestXPprCIBehPermGet” on page 148
“BestXPprProg” on page 111

BestXPprCIBehResultGet

Call

```
bx_errtype BestXPprCIBehResultGet(
      bx_handletype          handle,
      bxppr_behresulttype    resultprop,
      bx_int32               *pValue );
```

Description Gets the permutation result for the specific PPR completer-initiator behavior result property from the host.

CLI Equivalent `BestXPprCIBehResultGet prop=<resultprop>`

CLI Abbreviation `pprcibehresultget prop=<resultprop>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

resultprop Behavior result property to be obtained; see *"bxppr_behresulttype" on page 202*.

Output Parameters **pValue** Value of the behavior result property; *"bxppr_behresulttype" on page 202*.

See also –

BestXPprCIDefaultSet

Call

```
bx_errtype BestXPprCIDefaultSet( bx_handletype handle );
```

Description Sets all PPR completer-initiator properties on the host to default values.

To write PPR settings to the testcard, use the `BestXPprProg` call.

CLI Equivalent `BestXPprCIDefaultSet`

CLI Abbreviation `pprcidefaultset`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters **handle** Session identification.

See also
"BestXPprCIBehPermDefaultSet" on page 148
"BestXPprCIBehListDefaultSet" on page 145
"BestXPprProg" on page 111

PPR Reporting Functions

The following functions are used for PPR reporting:

Function	Action
“BestXPprReportWrite” on page 151	Generates a PPR report.
“BestXPprReportFile” on page 152	Writes the report to a file.
“BestXPprReportSet” on page 153	Sets report properties.
“BestXPprReportGet” on page 152	Gets report properties.

BestXPprReportWrite

Call

```
bx_errtype BestXPprReportWrite(
      bx_handletype    handle,
      bx_charptrtype   *reportStr );
```

Description

Generates a PPR report for the current settings. This function performs the permutations using the properties of initiator block variations, initiator behavior and target behavior, and reports the results.

To determine whether the CLI abbreviations are included in the report, use the `BestXPprReportSet` call.

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, you must free the memory space with the `BestXPprDelete` call.

CLI Equivalent

`BestXPprReportWrite`

The report string will be displayed in the CLI window.

CLI Abbreviation

`pprreportwrite`

Return Value

Error code; see *“bx_errtype” on page 176*.

Input Parameters

handle Session identification.

Output Parameters

reportStr Pointer to the report when the return of the function was successful.

See also

“BestXPprReportFile” on page 152
“BestXPprReportSet” on page 153
“BestXPprReportGet” on page 152

BestXPprReportFile

Call

```
bx_errtype BestXPprReportFile(
      bx_handletype    handle,
      bx_charptrtype   filename );
```

Description Generates a file that contains the PPR report for the current settings.

CLI Equivalent `BestXPprReportFile filename=<filename>`

CLI Abbreviation `pprreportfile file=<filename>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

filename Name of the file to which the report is to be written.

See also *“BestXPprReportWrite” on page 151*
“BestXPprReportSet” on page 153
“BestXPprReportGet” on page 152

BestXPprReportGet

Call

```
bx_errtype BestXPprReportGet(
      bx_handletype      handle,
      bxppr_reporttype   reportprop,
      bx_int32           *pValue );
```

Description Gets the PPR report property. The report property defines whether CLI abbreviations are included in the report.

CLI Equivalent `BestXPprReportGet reportprop=<reportprop>`

CLI Abbreviation `pprreportget prop=<reportprop>`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

reportprop Report property to be obtained; see *“bxppr_reporttype” on page 205*.

Output Parameters **pValue** Value of the report property; see *“bxppr_reporttype” on page 205*.

See also *“BestXPprReportSet” on page 153*

BestXPprReportSet

Call

```
bx_errtype BestXPprReportSet(
      bx_handletype      handle,
      bxppr_reporttype   reportprop,
      bx_int32           value );
```

Description Sets a PPR report property. The report property defines whether CLI abbreviations are included in the report.

CLI Equivalent `BestXPprReportSet reportprop=<reportprop> value=<value>`

CLI Abbreviation `pprreportset prop=<reportprop> val=<value>`

Return Value Error code; see *"bx_errtype" on page 176*.

Input Parameters

handle Session identification.

reportprop Report property to be set; see *"bxppr_reporttype" on page 205*.

value Value of the report property to be set; see *"bxppr_reporttype" on page 205*.

See also *"BestXPprReportGet" on page 152*

Error Handling

The application programming interface of the testcard provides several functions to handle errors.

Error functions can be used to query the error code and the error string of the last error that occurred in the specified session.

Error Functions

The following functions are used for error handling:

Function	Action
“BestXLastErrorGet” on page 157	Returns the last error.
“BestXLastErrorStringGet” on page 157	Returns the error string of the last error.
“BestXErrorStringGet” on page 156	Returns the error string of an error.

BestXErrorStringGet

Call

```
bx_charptrtype BestXErrorStringGet(
      bx_handletype     handle,
      bx_errtype        error,
      bx_charptrtype    *errptr );
```

Description Returns the error string of a certain error. The error string might contain placeholders for additional information. This is a generic function intended to find the error string for an arbitrary error code.

CLI Equivalent `BestXErrorStringGet`

CLI Abbreviation `errorstringget`

Return Value Pointer to the error string.

Input Parameters

handle Session identification.

error Error code; see *“bx_errtype” on page 176*.

Output Parameters

errptr Character pointer that points to the error string template.

See also

“BestXLastErrorGet” on page 157
“BestXLastErrorStringGet” on page 157

BestXLastErrorGet

Call

```
bx_errtype BestXLastErrorGet(
      bx_handletype   handle,
      bx_errtype      *error );
```

Description Returns the code of the last error that occurred within the specified session.

CLI Equivalent `BestXLastErrorGet`

CLI Abbreviation `lasterrget`

Return Value Error code; see *“bx_errtype” on page 176*.

Input Parameters **handle** Session identification.

Output Parameters **error** Error code; see *“bx_errtype” on page 176*. If no error occurred, BX_E_OK is returned.

See also *“BestXLastErrorStringGet” on page 157*
“BestXErrorStringGet” on page 156

BestXLastErrorStringGet

Call

```
bx_charptrtype BestXLastErrorStringGet( bx_handletype handle );
```

Description Reads the error string of the last error that occurred within the specified session. The error string is completely filled out and contains no place holders.

CLI Equivalent No CLI equivalent.

CLI Abbreviation No CLI abbreviation.

Return Value Pointer to the error string.

Input Parameters **handle** Session identification.

See also *“BestXLastErrorGet” on page 157*
“BestXErrorStringGet” on page 156

Type Definitions

All type definitions are listed in alphabetical order.

Type names are written with lowercase letters. In general, name begins with `bx_` and ends with `type`. Types that are used exclusively by Protocol Permutator and Randomizer functions begin with `bxppr_`.

bx_addrspacetype

prop	CLI Abbreviation	Description
BX_ADDRSPACE_MEM	mem	Data Memory Write or Data Memory Read is used as PCI-X command.
BX_ADDRSPACE_IO	io	IO Write or IO Read is used as PCI-X command.
BX_ADDRSPACE_CONFIG	config	Configuration space accesses are used.

bx_boardtype

NOTE Board and testcard have the same meaning.

<table>
<tr><th>prop
(CLI Abbreviation)</th><th>val</th><th>Description</th></tr>
<tr><td rowspan="3">BX_BOARD_RESPECTBIOS
(respectbios)</td><td colspan="2">Defines the behavior of the testcard at power up or reset with regard to the content of the configuration space.</td></tr>
<tr><td>default: 1</td><td>Only the fixed bits of the configuration space are preset to the values given by the user, all BIOS-owned bits (the read/write bits) are 0.</td></tr>
<tr><td>0</td><td>All values are set as given by the user. This is useful in systems without a BIOS or with no correct address assignment.
<b>Caution:</b> This may cause the system to hang or may even damage the system if several devices decode the same address.</td></tr>
<tr><td rowspan="3">BX_BOARD_BUSCLOCK
(busclock)</td><td colspan="2">This property is to be used in cases where the bus clocking rate is extremely slow (below 100 kHz). In these cases the synchronization between the clock domains within the PCI-X exerciser ASIC might not work as expected. You can then force synchronization.</td></tr>
<tr><td>default:
BX_BOARD_BUSCLOCK_DEF</td><td>Standard synchronization method. This will work in almost 100% of all cases.</td></tr>
<tr><td>BX_BOARD_BUSCLOCK_SLOW</td><td>The synchronization of the internal ASIC register is forced. This is <i>only</i> to be used with extremely slow bus speeds.</td></tr>
</table>

bx_cibehtype

prop (CLI Abbreviation)	Property Description	
	val	**Value Description**
BX_CIBEH_QUEUE (queue)	Selects the request queue from which the next (partial) completion is generated. If BX_CIBEH_QUEUE_NONE is used, requests will be accumulated in the request queues for out-of-order completion. **Caution:** If not all queues are selected, completion of requests parked in the queue that has not been selected is stalled forever. To avoid this, do not use BX_CIBEH_QUEUE_xWAIT (unless really needed), or ensure that at least one entry contains BX_CIBEH_QUEUE_NEXT or BX_CIBEH_QUEUE_AUTO or that all four queues are selected using BX_CIBEH_QUEUE_xSKIP.	
	default: BX_CIBEH_QUEUE_NEXT	The next queue is selected following the one selected previously. This gives every queue a chance to complete.
	BX_CIBEH_QUEUE_NONE	No queue is selected. This value can be used in conjunction with BX_CIBEH_REPEAT to avoid scheduling any completions for the specified number of behaviors. Incoming requests are then accumulated and can be completed later in any order.
	BX_CIBEH_QUEUE_AUTO	Any non-empty available queue is selected.
	BX_CIBEH_QUEUE_ASKIP, BX_CIBEH_QUEUE_BSKIP, BX_CIBEH_QUEUE_CSKIP, BX_CIBEH_QUEUE_DSKIP	Queue A/B/C/D is selected for the next completion transaction. If this queue is currently empty, the current behavior is skipped.
	BX_CIBEH_QUEUE_AWAIT, BX_CIBEH_QUEUE_BWAIT, BX_CIBEH_QUEUE_CWAIT, BX_CIBEH_QUEUE_DWAIT	Queue A/B/C/D is selected for the next completion transaction. If this queue is currently empty, execution waits until this queue gets a request. **Caution:** Be very careful using this option, because requests in other queues will not be completed if the selected queue does not receive a request. For programming a PCI-X-compliant behavior, do **not** use these options. A device is not supposed to make a completion dependent on the receipt of another transaction.
BX_CIBEH_ERRMESSAGE (errmessage)	Selects between a normal split completion transaction or write completion message and a user-defined split completion message (SCM). The SCM and SCE (split completion error) bits of the split completion-initiator attributes are controlled.	
	default: 0	SCE=0: Normal split completion with read data (SCM=0) or write completion message (SCM=1).
	1	SCE=1: Split completion error message (SCM=1). The message content is determined by BX_CIGEN_MESSAGE_AD.

prop (CLI Abbreviation)	Property Description	
	val	Value Description
BX_CIBEH_PARTITION (partition)	Limits the size of the (partial) completion transaction. The transaction may actually be shorter, if the request is satisfied earlier or the target terminates the transaction. In the latter case, another transaction is generated immediately to complete the intended (partial) completion.	
	default: BX_CIBEH_PARTITION_NO	The full byte count is transferred without disconnecting the completion.
	1 … 63	The completion is disconnected at every n-th allowable disconnect boundary (ADB) after the current start address, where n is in the range of 1 … 63.
BX_CIBEH_CONDSTART (condstart)	The conditional start flag defines whether the completion starts unconditionally or whether a conditional start pattern must have occurred.	
	default: BX_CIBEH_CONDSTART_NO	Unconditional start.
	BX_CIBEH_CONDSTART_ONCE1	After the exerciser has been started, block execution waits until the conditional start pattern 1 has occurred at least once.
	BX_CIBEH_CONDSTART_ONCE2	After the exerciser has been started, block execution waits until the conditional start pattern 2 has occurred at least once.
	BX_CIBEH_CONDSTART_WAIT1	After the end of the previous completer-initiator sequence, block execution waits until the conditional start pattern 1 has occurred.
	BX_CIBEH_CONDSTART_WAIT2	After the end of the previous completer-initiator sequence, block execution waits until the conditional start pattern 2 has occurred.
BX_CIBEH_DELAY (delay)	Clock delay before REQ# is asserted if a transaction is ready to be generated. That means no conditional start or wait-for-completion is enabled. The bus is reserved for the testcard.	
	1 … 65535 default: 1	Number of clock cycles that are inserted (a value of 1 introduces one idle cycle between transactions).
BX_CIBEH_STEPS (steps)	Number of address steps. That is the number of clock cycles between the assertion of GNT# and the assertion of FRAME#. According to the PCI-X specification, for configuration cycles, this number is ignored and always four address steps are inserted.	
	0 … 4 default: 2	Number of address steps.
	5, 6	For programming a PCI-X-compliant behavior, do **not** use these values.
BX_CIBEH_REQ64 (req64)	64-bit data transfer request. This property is not evaluated if it is used with a DWORD command. REQ64 will not be asserted for DWORD commands, regardless of the property setting.	
	default: 1	REQ64# will be asserted.
	0	REQ64# will not be asserted.
BX_CIBEH_RELREQ (relreq)	Defines the number of clock cycles after that REQ# is deasserted.	
	1 … 2047 default: 2047	Number of clock cycles after the address phase.
BX_CIBEH_REPEAT (repeat)	1 … 256 default: 1	Defines how often the current behavior is applied before the next behavior is used.

bx_cigentype

prop (CLI Abbreviation)	val	Description
BX_CIGEN_NUMBEH (numbeh)	1 … 256 default: 1	Defines how many complete-initiator behaviors are used before the first behavior is used again. Execution starts with behavior 0 and continues through BX_CIGEN_NUMBEH-1 and repeats.
BX_CIGEN_MESSAGE_AD (mesad)	32 bit field default: 0	Defines the content of the split completion message by putting the bits 19 … 31 on the AD bus during the data phase. Bits 0 … 11 are determined by the remaining byte count (determined by BX_RIBEH_BYTECOUNT). For generating a split completion message, see BX_CIBEH_ERRMESSAGE.

bx_ctbehtype

prop (CLI Abbreviation)	Property Description	
	val	Value Description
BX_CTBEH_DECSPEED (decspeed)	Defines the decode speed for the transaction.	
	BX_CTBEH_DECSPEED_A	Asserts DEVSEL# with speed A. This property works only up to 66 MHz. For 133 MHz, this value defaults to decode speed B.
	default: BX_CTBEH_DECSPEED_B	Asserts DEVSEL# with speed B.
	BX_CTBEH_DECSPEED_C	Asserts DEVSEL# with speed C.

prop (CLI Abbreviation)	Property Description	
	val	**Value Description**
BX_CTBEH_INITIAL (initial)	Specifies the initial target response. If the split response condition is true (see *"bx_ctsplittype" on page 166*), the property BX_CTBEH_SPLITLATENCY is applied instead. Wait cycles mentioned below start counting after the assertion of DEVSEL#.	
	default: BX_CTBEH_INITIAL_ACCEPT	The target accepts the data transfer or signals a split response after a minimum of the specified wait cycles. The number of wait cycles might actually be higher (but always within the limits of the spec), depending on history and direction of transfer. To comply with the spec, for write blocks, the number is rounded at runtime to an even number. The behavior of the target is non-PCI-X-compliant for BX_CTBEH_LATENCY values between 16 … 31.
	BX_CTBEH_INITIAL_SINGLE	The target signals a single data phase disconnect after the given number of wait cycles (values 0 … 15). The number of wait cycles might actually be higher (but always within the limits of the spec), depending on history and direction of transfer. For write blocks, the number is rounded at runtime to an even number (to comply with the spec). The behavior of the target is non-PCI-X compliant for BX_CTBEH_LATENCY values between 16 … 31.
	BX_CTBEH_INITIAL_RETRY	The target signals retry after a specified number of wait cycles (values 0 … 7). The behavior of the target is non-PCI-X-compliant for BX_CTBEH_LATENCY values between 8 … 31.
	BX_CTBEH_INITIAL_TABORT	The target signals a target abort after a specified number of wait cycles (values 0 … 7). The behavior of the target is non-PCI-X compliant for BX_CTBEH_LATENCY values between 8 … 31.
BX_CTBEH_ACK64 (ack64)	64-bit data transfer acknowledge.	
	default: 1	ACK64# will be asserted.
	0	ACK64# will not be asserted.
BX_CTBEH_LATENCY (latency)	3 … 34 default: 3	See BX_CTBEH_INITIAL for exact definition of the range.
BX_CTBEH_SUBSEQ (subseq)	Specifies the target response in subsequent data phases, which comes only into effect when the initial response was BX_CTBEH_INITIAL_ACCEPT.	
	default: BX_CTBEH_SUBSEQ_ACCEPT	Accepts all subsequent data phases. No target termination is signaled. BX_CTBEH_SUBSEQPHASE does not have a meaning in this case.
	BX_CTBEH_SUBSEQ_DISCONNECT	Signals a target disconnect in the selected data phase.
BX_CTBEH_SUBSEQPHASE (subseqphase)	0 … 2047 default: 0	See BX_CTBEH_SUBSEQ for an exact definition of the range.

prop (CLI Abbreviation)	Property Description	
	val	Value Description
BX_CTBEH_SPLITLATENCY (splitlatency)	When a split response is to be signaled as determined by the split response condition properties, this property defines the number of wait cycles before the split response is signaled. Otherwise property BX_CTBEH_INITIAL defines the initial behavior.	
	3 … 18 default: 3	A split response is signaled after the specified number of wait cycles. For programming a PCI-X-compliant behavior, do **not** use values in the range of 8 … 15.
BX_CTBEH_SPLITENABLE (splitenable)	Defines whether a split response is generated. To enable split response generation, the decoder must be set up accordingly (see *"Decoder Programming Functions" on page 92*).	
	default: 1	A split response will be generated.
	0	No split response will be generated.
BX_CTBEH_REPEAT (repeat)	1 … 65536 default: 1	Defines how often the current behavior is applied before the next behavior is used.

bx_ctgentype

prop	CLI Abbreviation	Values	Description
BX_CTGEN_NUMBEH	numbeh	1 … 256 default: 1	Defines how many target behaviors are used before the first one is used again.

bx_ctsplittype

prop (CLI Abbreviation)	Property Description	
	val	Value Description
BX_CTSPLIT_ADDRMASK_HI (addrmaskhi), BX_CTSPLIT_ADDRMASK_LO (addrmasklo)	Allows you to identify a request that will be given a split response, based on the address value shown in the address phase (see BX_CTSPLIT_ADDRVAL_xx).	
	0 … 0xffffffff default: 0xffffffff	If bit ‘n’ is set to ‘0’, the corresponding AD[n] signal is don’t care. If bit ‘n’ is set to ‘1’, the corresponding AD[n] signal must equal the corresponding bit in BX_CTSPLIT_ADDRVAL_xx.
BX_CTSPLIT_ADDRVAL_HI (addrvalhi), BX_CTSPLIT_ADDRVAL_LO (addrvallo)	0 … 0xffffffff default: 0	See BX_CTSPLIT_ADDRMASK_xx.
BX_CTSPLIT_CMDS (cmds)	Allows you to restrict split responses to only a subset of all 16 possible PCI-X commands.	
	16-bit field	If bit ‘N’ is set to ‘1’, a command whose encoding on C/BE equals ‘N’ can be split. Otherwise BX_CTBEH_INITIAL defines the initial target response.
	default: BX_CTSPLIT_CMDS_READBLOCK	With this predefined value, only memory read blocks can be split.
BX_CTSPLIT_DEC (dec)	Allows you to identify a request that will be given a split response, based on the address range (decoder) that has been accessed. If a split response is to be given, requests must always be claimed by one of the decoders.	
	BX_CTSPLIT_DEC_1	Address range number 1 (BAR 0 and BAR 1).
	default: BX_CTSPLIT_DEC_2	Address range number 2 (BAR 2 and BAR 3).
	BX_CTSPLIT_DEC_3	Address range number 3 (BAR 4 and BAR 5).
	BX_CTSPLIT_DEC_ANY	Any address range.
	BX_CTSPLIT_BAR_ANY	Any base address register.
	BX_CTSPLIT_BAR_0 BX_CTSPLIT_BAR_1 BX_CTSPLIT_BAR_2 BX_CTSPLIT_BAR_3 BX_CTSPLIT_BAR_4 BX_CTSPLIT_BAR_5	Base address register 0 … 5. **Note:** Use BX_CTSPLIT_BAR_0 ... 5 for IO decoders only. For memory decoders use BX_CTSPLIT_DEC_1 ... 3. Using BX_CTSPLIT_BAR_0 ... 5 for memory decoders generates unpredictable results.

prop (CLI Abbreviation)	Property Description	
	val	Value Description
BX_CTSPLIT_QUEUE (queue)	Selects a queue to which the request will be placed. If the requested queue is full, a retry is signaled instead of a split response. **Caution:** Make sure that all queues that are being filled get a chance to complete (see completer-initiator behavior property BX_CIBEH_QUEUE).	
	default: BX_CTSPLIT_QUEUE_A BX_CTSPLIT_QUEUE_B BX_CTSPLIT_QUEUE_C BX_CTSPLIT_QUEUE_D	Selects queue A/B/C/D.
	BX_CTSPLIT_QUEUE_NEXT	The next queue is selected following the one selected previously.
	BX_CTSPLIT_QUEUE_AUTO	Automatically selects any free queue.

bx_decproptype

prop (CLI Abbreviation)	Property Description	
	val	**Value Description**
BX_DECP_SIZE (size)	When the decoder size is larger than the size of the internal data resource (1MB for the data memory), the data resource is tiled.	
	Memory: 0, 7 … 63 IO: 0, 2 … 31 Exp ROM: 0, 11 … 24 defaults: Standard decoders: 12 Expansion ROM decoder: 0 Configuration space decoder: 1 Requester target decoder: 1	A value of 0 switches the decoder off. **Caution:** This performs NO boundary check! The testcard does not interrupt a burst at the boundary of the address range. Therefore the last QWORD should not be accessed.
BX_DECP_LOCATION (location)	Specifies the location for the requested address range. Changing the location automatically changes the size to default in order to prevent potential illegal combinations. Therefore, this property must be set first.	
	default: BX_DECP_LOCATION_MEM	Requests memory space (always 64-bit address). This always covers two adjacent BARs (an even (0,2,4) and the following odd BAR).
	BX_DECP_LOCATION_IO	Requests I/O space (32-bit address).
	BX_DECP_LOCATION_OFF	Defines a zero-sized (non-existent) decoder, where mask and value are zero.
BX_DECP_PREFETCH (prefetch)	0,1 default: 0	Defines if the memory is prefetchable.
BX_DECP_RESBASE (resbase)	0 … 2^{24} in steps of 4 default: 0	Internal resource base address for BX_DECP_RESOURCE.
BX_DECP_COMPARE (compare)	0, 1 default: 0	Switches on the data compare on for this decoder. Compare source is selected with BX_DECP_RESOURCE.
BX_DECP_RESSIZE (ressize)	2 … 24 default: 20	Only valid for STD (split transaction decoder) and Expansion ROM decoder.
BX_DECP_RESOURCE (resource)	Defines the data source for data transfer when the testcard is acting as a target.	
	default: BX_DECP_RESOURCE_MEM	Data memory.
	BX_DECP_RESOURCE_GEN	Data generator. Selection of the data generator is determined by the generic exerciser property BX_EGEN_DATAGEN.
	BX_DECP_RESOURCE_FLASH	Expansion ROM decoder flash memory. Can only be selected by the Expansion ROM decoder.

bx_dectype

prop	CLI Abbreviation	Description
BX_DEC_BAR0	bar0	Accesses Bar 0.
BX_DEC_BAR1	bar1	Accesses Bar 1.
BX_DEC_BAR2	bar2	Accesses Bar 2.
BX_DEC_BAR3	bar3	Accesses Bar 3.
BX_DEC_BAR4	bar4	Accesses Bar 4.
BX_DEC_BAR5	bar5	Accesses Bar 5.
BX_DEC_EXPROM	exprom	Accesses the Expansion ROM decoder.
BX_DEC_CONFIG	config	Accesses the Configuration Space decoder.
BX_DEC_RT	rtdec	Accesses the requester target decoder (split response).

bx_egentype

prop (CLI Abbreviation)	Value Description	
	val	Value Description
BX_EGEN_ERR_SOURCE (errsource)	Defines whether and from which resource an error is injected. The property BX_EGEN_ERR_NUM specifies the exact location within that resource. The check for the valid range is performed at programming time.	
	default: BX_EGEN_ERR_SOURCE_NONE	No error is injected.
	BX_EGEN_ERR_SOURCE_RIBLK	Error is injected from requester-initiator block memory. Valid range for BX_EGEN_ERR_NUM is: 0 … (BX_RIGEN_NUMBLK-1).
	BX_EGEN_ERR_SOURCE_RIBEH	Error is injected from requester-initiator behavior memory. Valid range for BX_EGEN_ERR_NUM is: 0 … (BX_RIGEN_NUMBEH-1).
	BX_EGEN_ERR_SOURCE_RTBEH	Error is injected from requester-target behavior memory. Valid range for BX_EGEN_ERR_NUM is: 0 … (BX_RTGEN_NUMBEH-1).
	BX_EGEN_ERR_SOURCE_DEC	Error is injected from decoder address range. Valid ranges for BX_EGEN_ERR_NUM are: 0 … 5 : BAR 0 … BAR 5 6 : expansion ROM decoder 7 : requester-target decoder 8 : configuration space decoder
	BX_EGEN_ERR_SOURCE_CTBEH	Error is injected from completer-target behavior memory. Valid range for BX_EGEN_ERR_NUM is: 0 … (BX_CTGEN_NUMBEH-1).
	BX_EGEN_ERR_SOURCE_CIBEH	Error is injected from completer-initiator behavior memory. Valid range for BX_EGEN_ERR_NUM is: 0 … (BX_CIGEN_NUMBEH-1).
BX_EGEN_ERR_NUM (errnum)	Defines where an error is injected. The valid range depends on the property BX_EGEN_ERR_SOURCE. The check is performed at programming time (when you call BestXExerciserProg).	
	default: 0	

prop (CLI Abbreviation)	Value Description	
	val	Value Description
BX_EGEN_ERR_PHASE (errphase)	Defines whether and when the possible exceptions are generated. See BX_EGEN_ERR_WRPAR, BX_EGEN_ERR_WRPAR64, BX_EGEN_ERR_PERR, BX_EGEN_ERR_SERR.	
	default: BX_EGEN_ERR_PHASE_ADDR	The exceptions are generated in the address phase.
	BX_EGEN_ERR_PHASE_ATTR	The exceptions are generated in the attribute phase.
	1 … 512 (1024 in 32 bit systems)	The exceptions are generated in the data phase with the specified number.
BX_EGEN_ERR_WRPAR (errwrpar)	A wrong parity (PAR) is generated one clock cycle after the phase determined by BX_EGEN_ERR_PHASE. If the testcard does not have this signal, either an error message is returned at programming time or this property is ignored. To determine if the error was injected successfully, you can query the exerciser status property BX_STAT_ERR_WRPAR.	
	default: 0	Parity is generated correctly.
	1	Parity is inverted.
BX_EGEN_ERR_WRPAR64 (errwrpar64)	A wrong parity (PAR64) is generated one clock cycle after the phase determined by BX_EGEN_ERR_PHASE. If the testcard does not have this signal, either an error message is returned at programming time or this property is ignored. To determine if the error was injected successfully, you can query the exerciser status property BX_STAT_ERR_WRPAR64.	
	default: 0	Parity is generated correctly.
	1	Parity is inverted.
BX_EGEN_ERR_PERR (errperr)	PERR is asserted two clocks after the data phase determined by BX_EGEN_ERR_PHASE. If the testcard does not have this signal, an error message is returned at programming time or this property is ignored. To determine whether the error was injected successfully, you can query the exerciser status property BX_STAT_ERR_PERR. The corresponding command register bit in configuration space must be set.	
	default: 0	PERR# is not asserted.
	1	PERR is asserted.
BX_EGEN_ERR_SERR (errserr)	SERR is asserted two clocks after the phase determined by BX_EGEN_ERR_PHASE for the duration of one clock. To determine whether the error was injected successfully, you can query the exerciser status property BX_STAT_ERR_SERR. The corresponding command register bit in configuration space must be set.	
	default: 0	SERR# is not asserted.
	1	SERR# is asserted.

prop (CLI Abbreviation)	Value Description	
	val	Value Description
BX_EGEN_INT_SOURCE (intsource)	Defines whether and when the involvement of the testcard in a particular transaction triggers the generation of interrupts. The property BX_EGEN_INT_NUM specifies the exact location within that resource. The check for the valid range is performed at programming time.	
	default: BX_EGEN_INT_SOURCE_NONE	No interrupt is asserted.
	BX_EGEN_INT_SOURCE_RIBLK	Interrupt is asserted at the start of the requester-initiator block with number BX_EGEN_INT_NUM (when BX_RIBEH_DELAY starts counting). Valid range for BX_EGEN_INT_NUM is: 0 … (BX_RIGEN_NUMBLK-1).
	BX_EGEN_INT_SOURCE_RIBEH	Interrupt is asserted at the start of the requester-initiator behavior with number BX_EGEN_INT_NUM (when BX_RIBEH_DELAY starts counting). Valid range for BX_EGEN_INT_NUM is: 0 … (BX_RIGEN_NUMBEH-1).
	BX_EGEN_INT_SOURCE_RTBEH	Interrupt is asserted at the start of the requester-target behavior with number BX_EGEN_INT_NUM. Valid range for BX_EGEN_INT_NUM is: 0 … (BX_RTGEN_NUMBEH-1).
	BX_EGEN_INT_SOURCE_DEC	Interrupt is asserted when the decoder address range BX_EGEN_INT_NUM is accessed. Valid ranges for BX_EGEN_INT_NUM are: 0 … 5 : BAR 0 … BAR 5 6 : expansion ROM decoder 7 : configuration space decoder 8 : requester-target decoder
	BX_EGEN_INT_SOURCE_CTBEH	Interrupt is asserted at the start of the completer-target behavior with number BX_EGEN_INT_NUM. Valid range for BX_EGEN_INT_NUM is: 0 … (BX_CTGEN_NUMBEH-1).
	BX_EGEN_INT_SOURCE_CIBEH	Interrupt is asserted at the start of the completer-initiator behavior with number BX_EGEN_INT_NUM (when BX_CIBEH_DELAY starts counting). Valid range for BX_EGEN_INT_NUM is: 0 … (BX_CIGEN_NUMBEH-1).
BX_EGEN_INT_NUM (intnum)	Defines where the interrupt is asserted. The valid range depends on the property BX_EGEN_INT_SOURCE. The check is performed at programming time (function call BestXExerciserProg).	
	default: 0	

prop (CLI Abbreviation)	Value Description	
	val	Value Description
BX_EGEN_INT_DELAYA (intdelaya), BX_EGEN_INT_DELAYB (intdelayb), BX_EGEN_INT_DELAYC (intdelayc), BX_EGEN_INT_DELAYD (intdelayd)	Defines whether and when interrupts INTA / INTB / INTC / INTD are asserted.	
	default: BX_EGEN_INT_DELAY_NO	No interrupt is asserted.
	1 … 65535	The interrupt is asserted n clocks after the event that triggers the interrupt, where n is in the range of 1 … 65535.
BX_EGEN_TRIG_SOURCE (trigsource)	Defines whether and when the involvement of the testcard in a particular transaction asserts the trigger output. The property BX_EGEN_TRIG_NUM specifies the exact location within that resource. The check for the valid range is performed at programming time.	
	default: BX_EGEN_TRIG_SOURCE_NONE	No trigger output is asserted.
	BX_EGEN_TRIG_SOURCE_RIBLK	Trigger output is asserted at the start of the requester-initiator block with number BX_EGEN_TRIG_NUM (when BX_RIBEH_DELAY starts counting). Valid range for BX_EGEN_TRIG_NUM is: 0 … (BX_RIGEN_NUMBLK-1).
	BX_EGEN_TRIG_SOURCE_RIBEH	Trigger output is asserted at the start of the requester-initiator behavior with number BX_EGEN_TRIG_NUM (when BX_RIBEH_DELAY starts counting). Valid range for BX_EGEN_TRIG_NUM is: 0 … (BX_RIGEN_NUMBEH-1).
	BX_EGEN_TRIG_SOURCE_RTBEH	Trigger output is asserted at the start of the requester-target behavior with number BX_EGEN_TRIG_NUM. Valid range for BX_EGEN_TRIG_NUM is: 0 … (BX_RTGEN_NUMBEH-1).
	BX_EGEN_TRIG_SOURCE_DEC	Trigger output is asserted when the decoder address range BX_EGEN_TRIG_NUM is accessed. Valid ranges for BX_EGEN_TRIG_NUM are: 0 … 5 : BAR 0 … BAR 5 6 : expansion ROM decoder 7 : configuration space decoder 8 : requester-target decoder
	BX_EGEN_TRIG_SOURCE_CTBEH	Trigger output is asserted at the start of the completer-target behavior with number BX_EGEN_TRIG_NUM. Valid range for BX_EGEN_TRIG_NUM is: 0 … (BX_CTGEN_NUMBEH-1).
	BX_EGEN_TRIG_SOURCE_CIBEH	Trigger output is asserted at the start of the completer-initiator behavior with number BX_EGEN_TRIG_NUM (when BX_CIBEH_DELAY starts counting). Valid range for BX_EGENTRIG_NUM is: 0 … (BX_CIGEN_NUMBEH-1).

prop (CLI Abbreviation)	Value Description	
	val	Value Description
BX_EGEN_TRIG_NUM (trignum)	Defines where the trigger output is asserted. The valid range depends on the property BX_EGEN_TRIG_SOURCE. The check is performed at programming time (function call BestXExerciserProg).	
	default: 0	
BX_EGEN_ARB (arb)	Defines how the internal arbiter decides if the next transaction is a requester-initiator transaction or a completer-initiator transaction. The properties BX_EGEN_ARBRI and BX_EGEN_ARBCI specify the selected arbitration algorithm.	
	default: BX_EGEN_ARB_AUTO	Requester-initiator and completer-initiator are selected one after the other. An empty queue is skipped.
	BX_EGEN_ARB_CONST	The arbiter takes a fixed number of requester-initiator transactions and then a fixed number of completer-initiator transactions. The number are defined by the BX_EGEN_ARBRI and BX_EGEN_ARBCI properties. **Caution:** It is possible to starve the card if the initiator is not set up correctly or no access to the card's target occurs.
	BX_EGEN_ARB_INCR	The arbiter increments the number of requester-initiator transactions and then the number of completer-initiator transactions. With every cycle, both numbers (BX_EGEN_ARBRI and BX_EGEN_ARBCI) are incremented by one. If one of the queues does not contain the required amount of transactions, the scheduler waits for 131072 clocks (2^{17}) before choosing the other queue. The incrementing of transactions takes place also if a queue is skipped. After incrementing up to 255, the counter restarts.
	BX_EGEN_ARB_RAND	For every cycle, random numbers for BX_EGEN_ARBRI and BX_EGEN_ARBCI are selected. The range is from 1 … 31. The arbiter takes the random number of requester-initiator transactions and then the selected number of completer-initiator transactions. If one of the queues does not contain the required amount of transactions, the scheduler waits for 131072 clocks (2^{17}) before choosing the other queue.
BX_EGEN_ARBRI (arbri)	1 … 254 default: 1	Number of requester-initiator transactions used for BX_EGEN_ARB (see above).
BX_EGEN_ARBCI (arbci)	1 … 254 default: 1	Number of completer-initiator transactions used for BX_EGEN_ARB (see above).

prop (CLI Abbreviation)	Value Description	
	val	Value Description
BX_EGEN_DATAGEN (datagen)	Defines the type of data generator that is used. This property controls the behavior if the data source 'data generator' was selected as requester-initiator block (BX_RIBLK_RESOURCE) or target decoder property (BX_DECP_RESOURCE).	
	default: BX_EGEN_DATAGEN_COUNTER	Counter with a programmable offset to the bus address. The counter creates a 64-bit wide data pattern, consisting of two count values from bit 0 … 22 and from 32 … 54. The remaining 18 bits can be preset with an arbitrary value or the initiator (requester-initiator and completer-initiator) identification to enable an easy identifier for any seen data in the system. In the case that the counter is chosen with an initiator identification, the data compare can be switched off for the 18 fixed bits to allow at least unidirectional data verification.
	BX_EGEN_DATAGEN_WALKING1	This data generator creates walking 1s.
	BX_EGEN_DATAGEN_WALKING0	This data generator creates walking 0s.
	BX_EGEN_DATAGEN_COUNTMIX	Counter with shuffled bits. A 64-bit wide pattern changes with a period of 2^{21} (looks like pseudo random).
	BX_EGEN_DATAGEN_GROUNDBOUNCE	This data generator creates 0x00000000 and 0xffffffff data patterns successively.
BX_EGEN_DATAFIX (datafix)	Defines the fixed value for the data generator. This property gets used only if BX_EGEN_DATAGEN_COUNTER is selected.	
	18 bit	Defines a fixed value for the data generator.
	default: BX_EGEN_DATAFIX_MASTERID	Uses the initiator ID as fixed value for the data generator.
BX_EGEN_DATASEED (dataseed)	20 bit default: 0	Starting seed or offset for the data generator.
BX_EGEN_PARTCOMP (partcomp)	Defines whether partial data compare is generally off or on. “Partial” refers to the fixed bit part when the data source 'data generator' was selected. See BX_EGEN_DATAGEN.	
	default: BX_EGEN_PARTCOMP_OFF	Switches partial data compare off.
	BX_EGEN_PARTCOMP_ON	Switches partial data compare on. **Caution:** BX_EGEN_PARTCOMP_ON restricts each data compare and not only the ones performed by the data generator.

bx_errtype

The software errors (codes beginning with “BX_E_”) are issued by the testcard software running on the control PC.

Error Code (Return Value)	Error String	Notes/Recommendations/Help
BX_E_ALIGN	Parameter <*parameter*> must be aligned to <*alignmt*>. Its value is <*value*>.	
BX_E_BAD_DECODER_NUMBER	No valid decoder number.	
BX_E_BAD_FILE_FORMAT	Bad format for file <*filename*>.	
BX_E_BAD_HANDLE	Bad handle. You may only use handles obtained from ”BestXOpen()”.	
BX_E_BAUDRATE	Could not set new baud rate.	
BX_E_CANNOT_CONNECT	The specified port <*port*> is unable to connect. Please check cable and connectors and try again.	Another reason may be that the card is busy, for example, with transferring data via the fast host interface.
BX_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.	

bx_obsruletype

Bit Pos.	rule	CLI Abbr.	Description	Reference
0	BX_OBSRULE_INITIATOR_0	initiator_0	An initiator can terminate a transaction with initiator abort (deassert FRAME# and IRDY#) 8 clocks after the address phase(s).	PCI-X Spec 2.11.1.2.
1	BX_OBSRULE_INITIATOR_1	initiator_1	If the target inserts wait states on Burst Write or Split Completion, the initiator must toggle between its first and second data values until the target asserts TRDY# (or terminates the transaction).	PCI-X Spec 1.10.2. Rule 5
2	BX_OBSRULE_INITIATOR_2	initiator_2	The initiator is required to terminate the transaction when the byte count is satisfied.	PCI-X Spec 1.10.2. Rule 6
3	BX_OBSRULE_INITIATOR_3	initiator_3	The initiator is permitted to disconnect a burst transaction (before the byte count is satisfied) only on an ADB.	PCI-X Spec 1.10.2. Rule 7
4	BX_OBSRULE_TARGET_0	target_0	Burst Write and Split Completion transactions must not be terminated with Split Response. All other target terminations are permitted.	PCI-X Spec 2.6.1.
5	BX_OBSRULE_TARGET_1	target_1	The target is permitted to signal Single Data Phase Disconnect only on the first data phase (with or without preceding Wait States).	PCI-X Spec 1.10.3. Rule 6
6	BX_OBSRULE_TARGET_2	target_2	The target is permitted to signal Split Response only on the first data phase (with or without preceding Wait States).	PCI-X Spec 2.11.2.4.
7	BX_OBSRULE_TARGET_3	target_3	When the target has signaled Disconnect on Next ADB, it must continue signaling this (or Target Abort) until the end of the transaction.	PCI-X Spec 2.11.2.2.
8	BX_OBSRULE_TARGET_4	target_4	The target deasserts DEVSEL#, STOP# and TRDY# one clock after the last data phase (if they are not already deasserted) and floats them one clock after that.	PCI-X Spec 1.10.3. Rule 8
9	BX_OBSRULE_TARGET_5	target_5	There must be an even number of target initial wait states for a burst write and Split Completion.	PCI-X Spec 2.9.
10	BX_OBSRULE_TARGET_6	target_6	If a PCI-X target signals Data Transfer (with or without preceding Wait States), the target is limited to disconnecting the transaction only on an ADB (until the byte count is satisfied).	PCI-X Spec 1.10.3. Rule 5
11	BX_OBSRULE_FRAME_0	frame_0	FRAME# cannot be deasserted unless IRDY# was asserted.	PCI Spec Appendix C, Rule 8c
12	BX_OBSRULE_FRAME_1	frame_1	When FRAME# has been deasserted, it cannot be reasserted during the same transaction.	PCI Spec Appendix C, Rules 8b and 8d
13	BX_OBSRULE_IRDY_0	irdy_0	IRDY# must be asserted two clocks after the attribute phase.	PCI-X Spec 1.10.2. Rule 3b
14	BX_OBSRULE_IRDY_1	irdy_1	Initiator Wait States are not permitted.	PCI-X Spec 1.10.2. Rule 3b

Bit Pos.	rule	CLI Abbr.	Description	Reference
15	BX_OBSRULE_IRDY_2	irdy_2	A transaction starts when FRAME# is asserted for the first time. IRDY# must not go low when FRAME# is high.	PCI Spec Appendix C, Rule 7 and 8c
16	BX_OBSRULE_IRDY_3	irdy_3	When IRDY# has been asserted, IRDY# must stay asserted until the end of transaction or till the target signals a termination.	PCI-X Spec 1.10.2. Rule 3b and PCI-X Spec 2.9.
17	BX_OBSRULE_TRDY_0	trdy_0	TRDY# must not be asserted before the attribute phase, it can only be asserted two or more clocks later.	PCI-X Spec 2.8. Table 2-6 and PCI Spec Appendix C Rule 14
18	BX_OBSRULE_TRDY_1	trdy_1	When TRDY# has been asserted, it must not be deasserted and reasserted during the same transaction (no subsequent wait states).	PCI-X Spec 1.10.3. Rule 4
19	BX_OBSRULE_DEVSEL_0	devsel_0	DEVSEL# must be asserted prior to the edge at which the target asserts TRDY#.	PCI-X Spec 2.8. and PCI-X Spec 2.9.1.
20	BX_OBSRULE_DEVSEL_1	devsel_1	DEVSEL# must not be asserted during a special cycle or if a reserved command has been used.	PCI-X Spec 2.4. and PCI-X Spec 2.7.3.
21	BX_OBSRULE_DEVSEL_2	devsel_2	DEVSEL# must not be asserted 1, 5 or more than 6 clocks after the address phase.	PCI-X Spec 2.8.
22	BX_OBSRULE_DEVSEL_3	devsel_3	After a Target asserts DEVSEL#, it cannot be deasserted until the last data phase has completed, except to signal Data Transfer, Wait States, Target Abort, Split Response, Retry and Single Data Phase Disconnect.	PCI-X Spec 1.10.3 Rule 3
23	BX_OBSRULE_DEVSEL_4	devsel_4	DEVSEL# must be deasserted one clock after last transfer.	PCI-X Spec 1.10.3 Rule 8
24	BX_OBSRULE_STOP_0	stop_0	STOP# must not be asserted without DEVSEL# being asserted, except RST# being asserted.	PCI-X Spec 1.10.1.Rule 12; PCI Spec Appendix C, Rule 14, Spec 6
25	BX_OBSRULE_LAT_0	lat_0	If the target signals Split Response, Target-Abort or Retry, the target must perform the corresponding action within eight clocks of the assertion of FRAME#.	PCI-X Spec 1.10.3 Rule 4
26	BX_OBSRULE_LAT_1	lat_1	If the target signals Single Data Phase Disconnect, Data Transfer or Disconnect on Next ADB, the target must perform the corresponding action within 16 clocks of the assertion of FRAME#.	PCI-X Spec 1.10.3 Rule 4
27	BX_OBSRULE_W64_0	w64_0	ACK64# may only be asserted if REQ64# was asserted before (ACK64# is a response to REQ64#).	PCI Spec 3.8.
28	BX_OBSRULE_W64_1	w64_1	A 64-bit initiator asserts REQ64# with the same timing as FRAME# to request a 64-bit data transfer. It deasserts REQ64# with FRAME# at the end of the transaction.	PCI-X Spec 2.12.3. Requirement 4

Bit Pos.	rule	CLI Abbr.	Description	Reference
29	BX_OBSRULE_W64_2	w64_2	If a 64-bit target is addressed by a transaction that has REQ64# asserted with FRAME#, the target asserts ACK64# with DEVSEL# to complete the transaction as a 64-bit target. It deasserts ACK64# with DEVSEL# at the end of the transaction.	PCI-X Spec 2.12.3.
30	BX_OBSRULE_W64_3	w64_3	REQ64# must not be used with special cycle or interrupt acknowledge command. Only burst transactions (memory commands other than Memory read DWORD) use 64-bit data transfers.	PCI-X Spec 2.4. and 2.7.
31	BX_OBSRULE_W64_4	w64_4	For DWORD Transactions, REQ64# must be deasserted.	PCI-X Spec 2.12.3. Requirement 2
32	BX_OBSRULE_PAR_0	par_0	PERR# may never be asserted three clocks after the address phase (or earlier in a transaction) or during a special cycle. During WRITE, PERR# may be asserted three clocks after IRDY#, during READ, PERR# may be asserted three clocks after TRDY#.	PCI Spec 3.8.2
33	BX_OBSRULE_PAR_1	par_1	AD[31::0] address parity error.	PCI Spec Appendix C, Rule 32 b
34	BX_OBSRULE_PAR_2	par_2	AD[63::32] address parity error.	PCI Spec Appendix C, Rule 32 c
35	BX_OBSRULE_PAR_3	par_3	AD[31::0] data parity error occurred but was not signaled.	PCI-X Spec 5.3.
36	BX_OBSRULE_PAR_4	par_4	AD[63::32] data parity error occurred but was not signaled.	PCI-X Spec 5.3.
37	BX_OBSRULE_PAR_5	par_5	AD[31::0] data parity error occurred.	PCI Spec Appendix C, Rule 32 b
38	BX_OBSRULE_PAR_6	par_6	AD[63::32] data parity error occurred.	PCI Spec Appendix C, Rule 32 c
39	BX_OBSRULE_SEM_0	sem_0	For I/O and DWORD Memory transactions, AD[1::0] and byte enable encoding must have a defined relationship. See table 2-2 in the PCI-X Spec.	PCI-X Spec 2.3, table 2-2
40	BX_OBSRULE_SEM_1	sem_1	Reserved commands are reserved for future use.	PCI-X Spec 2.4
41	BX_OBSRULE_SEM_2	sem_2	DAC is not allowed immediately after a DAC.	PCI Spec 3.9. and PCI-X Spec 2.12.1.
42	BX_OBSRULE_SEM_3	sem_3	During the data phases C/BE# bus must be driven high for all Burst Transactions except Memory Write.	PCI-X Spec 2.6.
43	BX_OBSRULE_SEM_4	sem_4	During a Dual Address Cycle, a 64-bit initiator has to drive the upper half of the address on AD[63::32] in the initial and in the second address phase.	PCI-X Spec 2.12.1.3 a i)
44	BX_OBSRULE_SEM_5	sem_5	In the second address phase of a Dual Address Cycle, AD [63:32] and AD [31:0] contain duplicate copies of the upper half of the address.	PCI-X Spec 2.12.1.3 a i)
45	BX_OBSRULE_SEM_6	sem_6	During a Dual Address Cycle, a 64-bit initiator has to drive the bus command on C/BE [7::4]# in the initial and the second address phase.	PCI-X Spec 2.12.1.3 a ii)

Bit Pos.	rule	CLI Abbr.	Description	Reference
46	BX_OBSRULE_SEM_7	sem_7	In the second address phase of a Dual Address Cycle, C/BE [7::4]# and C/BE [3::0]# contain duplicate copies of the transaction command.	PCI-X Spec 2.12.1.3 a ii)
47	BX_OBSRULE_SEM_8	sem_8	DWORD Transactions only support a single data phase.	PCI-X Spec 2.7.
48	BX_OBSRULE_SEM_9	sem_9	A initiator that supports 64-bit addressing must generate a SAC instead of a DAC when the upper 32 bits of the address are zero.	PCI Spec 3.9
49	BX_OBSRULE_SEM_10	sem_10	This rule detects a bus hang. If the bus does not get idle once within 4095 clocks, an error is registered.	Agilent Rule to detect potential deadlocks
50	BX_OBSRULE_SEM_11	sem_11	A initiator did not respond to a split transaction within 2^{20}-1 clocks.	Agilent Rule to detect potential deadlocks
51	BX_OBSRULE_SEM_12	sem_12	A delayed transaction (retries) has not been repeated within 2^{15} clocks.	PCI Spec 3.3.3.3.3
52	BX_OBSRULE_SEM_13	sem_13	A delayed transaction has not terminated within 2^{16} clocks.	Agilent Rule to detect potential deadlocks

bx_porttype

port	portnum	Description
BX_PORT_OFFLINE	Result of: *assumed* capabilities OR *assumed hardware* Values for *assumed hardware* are: BX_HW_E2929A, BX_HW_E2929A_DEEP, BX_HW_E2922A BX_HW_E2929B, BX_HW_E2929B_DEEP, BX_HW_E2922B	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 capability codes. **Example:** (BX_HW_E2929A | BX_CAPABILITY_64_BIT)
BX_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 BestXDevIdentifierGet 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*).

bx_resourcetype

resource	CLI Abbreviation	Description
BX_RESLOCK_EXERCISER	exe	Locks the exerciser.
BX_RESLOCK_OBSERVER	obs	Locks the observer.
BX_RESLOCK_ANALYZER	ana	Locks the trigger sequencer controls, the performance sequencers and counters and all pattern terms. This lock can only be activated with an E2922A/B testcard with option #100 (Analyzer piggy back card).
BX_RESLOCK_PERFORMANCE	per	Locks the performance sequencer.

bx_ribehtype

prop (CLI Abbreviation)	Value Description	
	val	Description
BX_RIBEH_QUEUE (queue)	Selects the requester-initiator queue from which the next sequence is generated. If the specified queue is empty (at the end of a test), another queue is selected.	
	BX_RIBEH_QUEUE_NEXT	The next queue after the last used queue is selected.
	default: BX_RIBEH_QUEUE_A	Queue A is selected.
	BX_RIBEH_QUEUE_B	Queue B is selected.
BX_RIBEH_TAG (tag)	Associates a tag to the sequence. Execution waits as long as the desired tag is in use. See also BX_STAT_TEST.	
	0 … 31	Associates the given tag to the sequence.
	default: BX_RIBEH_TAG_AUTO	Associates any free tag if possible.
BX_RIBEH_BYTECOUNT (bytecount)	1 … 4096 default: 4096	Generates a sequence with the given byte count, which is shown in the attribute phase. The desired byte count might actually be limited at runtime to the maximum byte count that is specified in the configuration space.
BX_RIBEH_DISCONNECT (disconnect)	Controls whether and how often the requester-initiator disconnects its current sequence.	
	default: BX_RIBEH_DISCONNECT_NO	The requester-initiator will not disconnect its sequence.
	1 … 32	The requester-initiator will always disconnect on every n-th allowable disconnect boundary (ADB), where n is in the range of 1 … 32.
BX_RIBEH_DELAY (delay)	Clock delay before REQ# is asserted if a transaction is ready to be generated. That means, no wait for a conditional start pattern or a free tag or the completion of all outstanding requests.	
	1 … 65535 default: 1	Number of clock cycles that are inserted—a value of 1 introduces one idle cycle between transactions.
BX_RIBEH_STEPS (steps)	Number of address steps. That is, the number of clock cycles between the assertion of GNT# and the assertion of FRAME# in addition to two clock cycles, which are designed into the register-to-register interface of PCI-X. According to the PCI-X specification, this property is ignored for configuration cycles and always four address steps are inserted.	
	0 … 4 default: 0	Number of address steps
	5, 6	For programming a PCI-X-compliant behavior, do **not** use these values.

prop **(CLI Abbreviation)**	**Value Description**	
	val	**Description**
BX_RIBEH_REQ64 (req64)	64-bit data transfer request. This property is not evaluated if it is used with a DWORD command. REQ64 will not be asserted for DWORD commands, regardless of the property setting.	
	default: 1	REQ64# will be asserted.
	0	REQ64# will not be asserted.
BX_RIBEH_RELREQ (relreq)	Defines the number of clock cycles after that REQ# is deasserted.	
	1 … 2047 default: 2047	Number of clock cycles after the address phase.
BX_RIBEH_REPEAT (repeat)	Defines how often the current behavior is applied before the next behavior is used.	
	1 … 256 default: 1	Number of repeats.
BX_RIBEH_SKIP (skip)	Controls the address calculations for subsequent transfers (see BX_RIBEH_REPEAT).	
	default: BX_RIBEH_SKIP_NO	No skip-register is selected. Data is transferred into a contiguous memory area.
	1 … 7	A skip-register is selected. The value of the skip-register is added to the next address phase.

bx_riblktype

prop (CLI Abbreviation)	Property Description	
	val (CLI Abbreviation)	Value Description
BX_RIBLK_BUSADDR_LO (busaddrlo)	32-bit value default: 0	Lower 32 bits of the bus address.
BX_RIBLK_BUSADDR_HI (busaddrhi)	32-bit value default: 0	Upper 32 bits of the bus address.
BX_RIBLK_CONDSTART (condstart)	The conditional start flag defines whether the requester-initiator block execution starts unconditionally or whether a conditional start pattern must have occurred.	
	default: BX_RIBLK_CONDSTART_NO	Unconditional start.
	BX_RIBLK_CONDSTART_ONCE1	After the exerciser has been started, block execution waits until the conditional start pattern 1 has occurred at least once.
	BX_RIBLK_CONDSTART_ONCE2	After the exerciser has been started, block execution waits until the conditional start pattern 2 has occurred at least once.
	BX_RIBLK_CONDSTART_WAIT1	After the end of the previous requester-initiator sequence, block execution waits until the conditional start pattern 1 has occurred.
	BX_RIBLK_CONDSTART_WAIT2	After the end of the previous requester-initiator sequence, block execution waits until the conditional start pattern 2 has occurred.
BX_RIBLK_BYTEN (byten)	0000\b … 1111\b default: 0	Values for the byte enables. The values are shown on C/BE[3::0] and C/BE[7::4] during *all* data phases of a block that is written. For a block that is read, this property is ignored. At the beginning and at the end of a sequence, the byte enables are masked. The mask is determined by the start address and the byte count.
BX_RIBLK_INTADDR (intaddr)	0x00000 … 0xfffff default: 0	Internal address of the testcard's internal data memory. The values must have the same QWORD alignment as BX_RIBLK_BUSADDR_LO.
BX_RIBLK_NUMBYTES (numbytes)	1 … 0xffffffff default: 1	Total number of bytes transferred in this logical block transfer. It is possible to cross the 4 GB address boundaries.

prop (CLI Abbreviation)	Property Description	
	val (CLI Abbreviation)	Value Description
BX_RIBLK_BUSCMD (buscmd)	Bus command seen on C/BE [3::0] in the address phase. **Note:** No special cycles can be generated.	
	BX_RIBLK_BUSCMD_INTACK	Interrupt acknowledge.
	BX_RIBLK_BUSCMD_IO_READ	I/O Read.
	BX_RIBLK_BUSCMD_IO_WRITE	I/O Write.
	BX_RIBLK_BUSCMD_RESERVED_4	For programming a PCI-X-compliant behavior, do **not** use this command. It will automatically be replaced by Memory Read DWORD.
	BX_RIBLK_BUSCMD_RESERVED_5	For programming a PCI-X-compliant behavior, do **not** use this command. It will automatically be replaced by Memory Write.
	default: BX_RIBLK_BUSCMD_MEM_READDWORD	Memory Read DWORD.
	BX_RIBLK_BUSCMD_MEM_WRITE	Memory Write.
	BX_RIBLK_BUSCMD_RESERVED_8 (reserved8)	For programming a PCI-X-compliant behavior, do **not** use this command. It will automatically be replaced by memreadblock.
	BX_RIBLK_BUSCMD_RESERVED_9 (reserved9)	For programming a PCI-X-compliant behavior, do **not** use this command. It will automatically be replaced by memwriteblock.
	BX_RIBLK_BUSCMD_CONFIG_READ	Configuration Read.
	BX_RIBLK_BUSCMD_CONFIG_WRITE	Configuration Write.
	BX_RIBLK_BUSCMD_SPLIT	Split completion. For programming a PCI-X-compliant behavior, do **not** use this command. It will automatically be replaced by memwrite. A split completion cannot be a request, it can only be a reaction to a request.
	BX_RIBLK_BUSCMD_MEM_READBLOCK	Memory Read Block.
	BX_RIBLK_BUSCMD_MEM_WRITEBLOCK	Memory Write Block.
BX_RIBLK_COMPLETION (completion)	Defines whether all outstanding requests have been completed before the block can be scheduled. **Note:** A new tag is always used only when no request with the same tag is pending.	
	default: 0	Block execution does not wait.
	1	Block execution waits until all outstanding requests have been completed.
BX_RIBLK_RELAXORDER (relaxorder)	0, 1 default: 1	Relaxed ordering bit that is shown in the attribute phase.
BX_RIBLK_NOSNOOP (nosnoop)	0, 1 default: 0	Bit showing that no snoop will be done. It will be shown in the attribute phase.
BX_RIBLK_RESERVED31 (reserved31)	Controls the value of AD[31] in the attribute phase.	
	0,1 default: 0	For programming a PCI-X-compliant behavior, do **not** set this property to 1.

prop (CLI Abbreviation)	Property Description	
	val (CLI Abbreviation)	Value Description
BX_RIBLK_QUEUE (queue)	Selects the requester-initiator queue into which the block is put. When all blocks are put into the same queue, they are executed one after the other.	
	default: BX_RIBLK_QUEUE_A	Queue A is selected.
	BX_RIBLK_QUEUE_B	Queue B is selected.
	BX_RIBLK_QUEUE_AUTO	Any free queue is selected automatically.
BX_RIBLK_RESOURCE (resource)	Defines the data source for data leaving the testcard and entering the testcard.	
	default: BX_RIBLK_RESOURCE_DATAMEM	Data memory.
	BX_RIBLK_RESOURCE_DATAGEN	Data generator. The type of the data generator can be determined with the generic exerciser property BX_EGEN_DATAGEN.
BX_RIBLK_DATACMP (datacmp)	Switches data compare on or off.	
	BX_RIBLK_DATACMP_ON	Data compare is switched on. The data will be compared with the internal data source (specified by BX_RIBLK_RESOURCE). The content of the data memory will not be changed.
	default: BX_RIBLK_DATACMP_OFF	Data compare is switched off.

bx_rigentype

prop (CLI Abbreviation)	Property Description	
	val	Value Description
BX_RIGEN_NUMBLK (numblk)	1 … 256 default: 1	Defines the number of logical block transfers that are executed.
BX_RIGEN_REPEATBLK (repeatblk)	Defines how often the programmed series of requester-initiator blocks is executed.	
	default: 1	The series of programmed block transfers is executed once.
	2 … 0xffffffff	Number of executions of the series of programmed block transfers.
	BX_RIGEN_REPEATBLK_INFINITE	Defines an infinite number of repeats until the requester-initiator stop command is executed or another termination condition is met.
BX_RIGEN_NUMBEH (numbeh)	1 … 256 default: 1	Defines how many requester-initiator behaviors are used before the first one is used again.
BX_RIGEN_TABORT (tabort)	Controls the requester-initiator behavior in the event of a target abort.	
	default: BX_RIGEN_TABORT_STOP	Execution stops immediately.
	BX_RIGEN_TABORT_SKIP	The current transaction is skipped and execution continues with the following sequence.
BX_RIGEN_IABORT (iabort)	Controls the requester-initiator behavior in the event of a initiator abort.	
	default: BX_RIGEN_IABORT_STOP	Execution stops immediately.
	BX_RIGEN_IABORT_SKIP	The current transaction is skipped and execution continues with the following sequence.
BX_RIGEN_SKIPREG1 (skipreg1), BX_RIGEN_SKIPREG2 (skipreg2), BX_RIGEN_SKIPREG3 (skipreg3), BX_RIGEN_SKIPREG4 (skipreg4), BX_RIGEN_SKIPREG5 (skipreg5), BX_RIGEN_SKIPREG6 (skipreg6), BX_RIGEN_SKIPREG7 (skipreg7)	0 … 1023 default: 0	Values for the seven skip-registers. **Note:** Values MUST be QWORD aligned. A skip register is selected with BX_RIBEH_SKIP. In case of repeated transfers with the same behavior (controlled by BX_RIBEH_REPEAT), the skip-register value is added to the address. This results in gaps the size of the skip-register in the target memory area. Using skip values does not change the amount of transferred data. The internal address is *not* incremented. Only the data generator (dependent on the external bus address) follows the skipped address.

bx_rtbehtype

prop (CLI Abbreviation)	Property Description	
	val	**Value Description**
BX_RTBEH_DECSPEED (decspeed)	Defines the decode speed for this transaction.	
	BX_RTBEH_DECSPEED_A	Asserts DEVSEL# with speed A. **Note:** This property works only up to 66 MHz. For 133 MHz and 100 MHz, this value defaults to decode speed B.
	default: BX_RTBEH_DECSPEED_B	Asserts DEVSEL# with speed B.
BX_RTBEH_ACK64 (ack64)	64-bit data transfer acknowledge. This property is constant for a whole sequence.	
	default: 1	ACK64# will be asserted in response to a REQ64# assertion.
	0	ACK64# will not be asserted.
BX_RTBEH_INITIAL (initial)	Specifies the initial target response. The value of this property specifies how the value of BX_RTBEH_LATENCY is to be interpreted.	
	default: BX_RTBEH_INITIAL_ACCEPT	The target accepts the data transfer after the given number of latency cycles. If the number of wait cycles turns out to be odd, the next lower even number is chosen.
	BX_RTBEH_INITIAL_SINGLE	The target signals a single data phase disconnect after the given number of latency cycles (values 3 … 19). The behavior of the requester target is non-PCI-X compliant for BX_RTBEH_LATENCY values of 19 … 34.
	BX_RTBEH_INITIAL_RETRY	The target signals retry after the given number of latency cycles (values 3 … 10). The behavior of the requester target is non-PCI-X compliant for BX_RTBEH_LATENCY values of 11 … 34.
	BX_INITIAL_TABORT	The target signals a target abort after the given number of latency cycles (values 3 … 10). The behavior of the requester target is non-PCI-X compliant for BX_RTBEH_LATENCY values of 11 … 34.
BX_RTBEH_LATENCY (latency)	3 … 34 default: 3	Number of initial latency clocks. Counting starts with first assertion of FRAME#. For further information, see BX_RTBEH_INITIAL.

prop (CLI Abbreviation)	Property Description	
	val	Value Description
BX_RTBEH_SUBSEQ (subseq)	Specifies the target response in subsequent data phases—initial response must be BX_INITIAL_ACCEPT.	
	default: BX_RTBEH_SUBSEQ_ACCEPT	Accepts all subsequent data phases. No target termination is signaled.
	BX_RTBEH_SUBSEQ_DISCONNECT + <1…1024>	Signals a target disconnect in the selected data phase. For programming a PCI-X-compliant behavior, do **not** use this option.
BX_RTBEH_SUBSEQPHASE (subseqphase)	0 … 2047 default: 0	See BX_RTBEH_SUBSEQ for exact description of the range.
BX_RTBEH_REPEAT (repeat)	1 … 65536 default: 1	Defines how often the current behavior is applied before the next behavior is used.

bx_rtgentype

prop	CLI Abbreviation	val	Description
BX_RTGEN_NUMBEH	numbeh	1 … 256 default: 1	Defines how many requester-target behaviors are used before the first one is used again.

bx_signaltype

The following table shows the types of signals, their capabilities and where they can be applied. The “signal type” column of the tables in the subsequent table 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, bus address or byte enables.
list	For multiple-bit signals whose value is encoded in multiple bit sequences, you can determine these values.	Single bits have no meaning (for example, commands). The elements of each list are numbered, for example, command ”I/O Read” is numbered 2. Only numeric values may be used.

The following table shows all signals, their signal type and for which pattern they can be used. It also indicates which signal is visible in the trace memory, and the bit width. Property names that are written in upper case are bus signals. For internal signals, lower case is used.

NOTE Bit positions written in parentheses are valid for gap mode only.

					Pattern Type		
Property Name	**Pattern and Trace Memory Syntax**	**Width (bit)**	**Signal Type**	**Trace Memory (bit pos.)**	**Bus Pattern**	**Observer Pattern**	**Error Pattern**
BX_SIG_AD64	AD64	32	10X vector	31::0	Yes	-	-
BX_SIG_AD32	AD32	32	10X vector	63::32	Yes	-	-
BX_SIG_FRAME	FRAME	1	10X	80	Yes	-	-
BX_SIG_IRDY	IRDY	1	10X	81	Yes	-	-
BX_SIG_TRDY	TRDY	1	10X	82	Yes	-	-
BX_SIG_DEVSEL	DEVSEL	1	10X	83	Yes	-	-
BX_SIG_IDSEL	IDSEL	1	10X	84	Yes	-	-
BX_SIG_STOP	STOP	1	10X	85	Yes	-	-
BX_SIG_REQ	REQ	1	10X	86	Yes	-	-
BX_SIG_GNT	GNT	1	10X	87	Yes	-	-
BX_SIG_PERR	PERR	1	10X	88	Yes	-	-
BX_SIG_SERR	SERR	1	10X	89	Yes	-	-
BX_SIG_PAR	PAR	1	10X	90	Yes	-	-
BX_SIG_RST	RST	1	10X	91	Yes	-	-
BX_SIG_INTA	INTA	1	10X	95	Yes	-	-
BX_SIG_INTB	INTB	1	10X	94	Yes	-	-
BX_SIG_INTC	INTC	1	10X	93	Yes	-	-
BX_SIG_INTD	INTD	1	10X	92	Yes	-	-
BX_SIG_CBE3_0	CBE3_0	4	10X vector	67::94	Yes	-	-
BX_SIG_CBE7_4	CBE7_4	4	10X vector	71::68	Yes	-	-
BX_SIG_LOCK	LOCK	1	10X	72	Yes	-	-
BX_SIG_PAR64	PAR64	1	10X	73	Yes	-	-
BX_SIG_REQ64	REQ64	1	10X	74	Yes	-	-
BX_SIG_ACK64	ACK64	1	10X	75	Yes	-	-
BX_SIG_trigio0	trigio0	1	10X	76	Yes	-	-
BX_SIG_trigio1	trigio1	1	10X	77	Yes	-	-
BX_SIG_trigio2	trigio2	1	10X	78	Yes	-	-
BX_SIG_trigio3	trigio3	1	10X	79	Yes	-	-
BX_SIG_bstate[a]	bstate	4	list	115::112	-	yes	-
BX_SIG_decode[a]	decode	3	list	118::116	-	yes	-

					Pattern Type		
Property Name	Pattern and Trace Memory Syntax	Width (bit)	Signal Type	Trace Memory (bit pos.)	Bus Pattern	Observer Pattern	Error Pattern
BX_SIG_term[a]	term	3	list	121::119	-	yes	-
BX_SIG_xact_cmd[a]	xact_cmd	4	list	125::122	-	yes	-
BX_SIG_xact_tran64	xact_tran64	1	10X	126	-	yes	-
BX_SIG_ri_act	ri_act	1	10X	127	-	yes	-
BX_SIG_ct_act	ct_act	1	10X	96	-	yes	-
BX_SIG_ci_act	ci_act	1	10X	97	-	yes	-
BX_SIG_rt_act	rt_act	1	10X	98	-	yes	-
BX_SIG_ri_done	ri_done	1	10X	99	-	yes	-
BX_SIG_proterr	proterr	1	10X	100	-		yes
BX_SIG_dcomperr	dcomperr	1	10X	101	-		yes
BX_SIG_spliterr	spliterr	1	10X	102	-		yes
BX_SIG_c_rule0	c_rule0	1	10X	103	-	yes	-
BX_SIG_c_rule1	c_rule1	1	10X	104	-	yes	-
BX_SIG_ri_split_pending	ri_split_pending	1	10X	107	-	yes	-
BX_SIG_pcix_men	pcix_men	1	10X	108	-	-	-
BX_SIG_ad_act	ad_act	2	-	110::109	-	-	-
BX_SIG_gap	-	1	-	111, (111)	-	-	-
BX_SIG_gap_count	-	32	-	(31::0)	-	-	-
-	addr_phase	1	-	-	yes	-	-
BX_SIG_addr_phase_occ	addr_phase_occ	1	-	(32)	-	-	-
-	xact_attr_ad	32	10X vector	-	-	yes	-
-	xact_attr_cbe	4	10X vector	-	-	yes	-
-	xact_dac	1	10X	-	-	yes	-
-	xact_lock	1	10X	-	-	yes	-
-	xact_trigio0	1	10X	-	-	yes	-
-	xact_trigio1	1	10X	-	-	yes	-
-	xact_trigio2	1	10X	-	-	yes	-
-	xact_trigio3	1	10X	-	-	yes	-

[a] For more information, see *“Values of List Signals” on page 193*.

Values of List Signals

The following table shows the values of the following list signals:

- BX_SIG_decode
- BX_SIG_term
- BX_SIG_xact_cmd
- BX_SIG_bstate (PCI-X mode only)
- BX_SIG_bstate (PCI mode only)

	Description	
signal	Values	Value Description
BX_SIG_decode	Decode speed of the current transaction. Once defined, the decode speed is valid for the entire transaction.	
	0	There is no started transaction or a started transaction has not yet been claimed by a target. This states is left if a target claims a transaction by asserting DEVSEL#.
	1	The current transaction is being decoded decoded with decode speed A.
	2	The current transaction is being decoded with decode speed B.
	3	The current transaction is being decoded with decode speed C.
	4	The current transaction is being decoded with subtractive decode speed.
	5	The current transaction is being decoded slower than a subtractive decoder decodes.
BX_SIG_term	0	This data phase is not the end of a transaction.
	1	This transaction is terminated by the initiator with the initiator abort protocol.
	2	This transaction is terminated by the target, which signals split response.
	3	This transaction is terminated by the target with the target abort protocol.
	4	This transaction is terminated by the target, which signals single data phase disconnect.
	5	This transaction is terminated by the target, which signals retry.
	6	This transaction is terminated by the target, which signals disconnect on next ADB.
	7	This transaction has not been terminated and finished successfully.

signal	Description	
	Values	**Value Description**
BX_SIG_xact_cmd	Command used for the current transaction. This information is valid between the address phase and the end of the transaction. In a dual address cycle it shows “DAC” in the first address cycle, and the bus command used in the second address cycle.	
	0	Interrupt Acknowledge
	1	Special Cycle
	2	I/O Read
	3	I/O Write
	4	Reserved
	5	Reserved
	6	Memory Read DWord
	7	Memory Write
	8	Alias to Memory Read Block
	9	Alias to Memory Write Block
	A	Configuration Read
	B	Configuration Write
	C	Split Completion
	D	Dual Address Cycle
	E	Memory Read Block
	F	Memory Write Block
BX_SIG_bstate (PCI-X mode only)	Values of the bus state signal in case the bus runs in **PCI-X** mode.	
	0	After reset, this state is active until an idle state (FRAME# and IRDY# deasserted) is detected.
	1	The bus is idle (FRAME# and IRDY# deasserted).
	2	The first half of a dual address cycle.
	3	Address phase of a single address cycle.
	4	The second half of a dual address cycle.
	5	One clock following the address phase. This phase provides further information about the transaction.
	6	One clock where the target claims the transaction by asserting DEVSEL#.
	7	A target has accepted the transfer (asserted DEVSEL#), however, this target has not yet asserted TRDY#. **Note:** Once asserted, TRDY# must not be deasserted and reasserted within the same transaction.
	8	Currently a data transfer is active (both IRDY# and TRDY# are asserted).
	9	One clock when FRAME# and IRDY# are still asserted, while TRDY# and DEVSEL# are both deasserted (only for DWORD transactions).
	A	One or more clocks between attribute and target respond.
	B	One clock after termination without continuing transfer (that is target abort).

signal	Description	
	Values	Value Description
BX_SIG_bstate (PCI mode only)	Values of the bus state signal in case the bus runs in **PCI** mode.	
	0	After reset, this state is active until an idle state (FRAME# and IRDY# deasserted) is detected.
	1	The bus is idle (FRAME# and IRDY# deasserted).
	2	The first half of a dual address cycle.
	3	Address phase of a single address cycle.
	4	The second half of a dual address cycle.
	5	The address phase has been completed, however, a target has not yet claimed the access.
	6	A target has accepted the transfer (asserted DEVSEL#). The target, the initiator or both are inserting wait cycles. If IRDY# is deasserted, the waits are inserted by the initiator, if TRDY# is deasserted, they are inserted by the target.
	7	Currently a data transfer is active (both IRDY# and TRDY# are asserted).

bx_sizetype

size	CLI Abbreviation
BX_SIZE_BYTE	byte
BX_SIZE_WORD	word
BX_SIZE_DWORD	dword

bx_statustype

NOTE Values that are indicated by “initial” are cleared state values.

	Property Description		
prop (CLI Abbreviation)	val	Value Description	Clearing the Status
BX_STAT_CMP_FAIL (fail)	Reports if a data compare error has been detected.		Cleared automatically with a call to BestXExerciserRun or with BestXStatusClear and property BX_STAT_CMP_FAIL.
	0 (initial)	No data compare error was detected.	
	1	A data compare error error was detected.	
BX_STAT_CMP_CMD (cmd)	4-bit value	Returns the bus command of a data compare error.	
BX_STAT_CMP_ADDR_HI (addrhi), BX_STAT_CMP_ADDR_LO (addrlo)	32-bit value	Returns the bus address of a data compare error.	
BX_STAT_CMP_EXTCMD (extcmd)	4-bit value	Returns the extended bus command of a data compare error.	
BX_STAT_CMP_ATTR_LO (attrlo)	32-bit value	Returns the attributes of a data compare error.	
BX_STAT_CMP_ACTUAL_HI (acthi), BX_STAT_CMP_ACTUAL_LO (actlo)	32-bit value	Returns the actual data of a data compare error.	
BX_STAT_CMP_REF_HI (refhi), BX_STAT_CMP_REF_LO (reflo)	32-bit value	Returns the reference data of a data compare error.	
BX_STAT_ERR_PERR (perr)	0 (initial)	PERR has not been asserted.	Cleared automatically with a call to BestXStatusClear and property BX_STAT_ERR_PERR.
	1	PERR has been asserted. This is equivalent to the PERR status bit in the configuration space.	
BX_STAT_ERR_SERR (serr)	0 (initial)	SERR has not been asserted.	Cleared automatically with a call to BestXStatusClear and property BX_STAT_ERR_SERR.
	1	SERR has been asserted. This is equivalent to the SERR status bit in the configuration space.	
BX_STAT_ERR_WRPAR (wrpar)	0 (initial)	PAR has not been inverted.	Cleared automatically with a call to BestXStatusClear and BX_STAT_ERR_WRPAR or BX_STAT_ERR_WRPAR64.
	1	PAR has been inverted.	
BX_STAT_ERR_WRPAR64 (wrpar64)	0 (initial)	PAR64 has not been inverted.	
	1	PAR64 has been inverted.	

prop (CLI Abbreviation)	Property Description		Clearing the Status
	val	Value Description	
BX_STAT_IABORT (iabort)	0 (initial)	No initiator abort has occurred.	Cleared automatically with a call to BestXExerciserRun or with BestXStatusClear and BX_STAT_IABORT or BX_STAT_TABORT.
	1	A initiator abort has occurred.	
BX_STAT_TABORT tabort)	0 (initial)	No target abort has occurred.	
	1	A target abort has occurred.	
BX_STAT_SPLITFAIL (splitfail)	Returns whether the requester target has received a split-completion error message.		Cleared automatically with a call to BestXExerciserRun or with BestXStatusClear and BX_STAT_SPLITFAIL.
	0 (initial)	No split-completion error message was received.	
	1	A split-completion error message was received.	
BX_STAT_INTA (inta), BX_STAT_INTB (intb), BX_STAT_INTC (intc), BX_STAT_INTD (intd)	0 (initial)	No interrupt has been asserted.	The status can be cleared with BestXStatusCLear and property BX_STAT_INT_xx.
	1	Interrupt A/B/C/D has been asserted as part of a test; see exerciser properties BX_EGEN_INT_x.	

prop (CLI Abbreviation)	Property Description		Clearing the Status
	val	Value Description	
BX_STAT_TEST (test)	Returns a 32-bit status value with the following bit masks. **Note:** The values BX_STAT_TEST_WAITING, BX_STAT_TEST_STOPPED and BX_STAT_TEST_RUNNING are exclusive. That means always exactly one of these values is set.		A clear has no effect.
	BX_STAT_TEST_RUNNING	'0': After power on and after the requester-initiator has successfully finished the programmed blocks.	
		'1': A test is currently running. This includes waiting for events such as conditional start or waiting for a completion message.	
	BX_STAT_TEST_WAITING	'1': The initiator (requester-initiator or completer-initiator) is waiting for the conditional start pattern.	
	BX_STAT_TEST_DONE	'1': The requester-initiator has finished the series of programmed blocks at the bus. This does not include waiting for all outstanding completions.	This bit is cleared by calls to BestXExerciserStop or BestXExerciserRun.
	BX_STAT_TEST_STOPPED	'1': The initiator (requester-initiator or completer-initiator) is stopped.	A clear has no effect.
	BX_STAT_TEST_COMPLETION	The requester-target is waiting for outstanding split completions.	
	BX_STAT_TEST_IABORT	Defines whether an initiator abort occurred.	
	BX_STAT_TEST_TABORT	Defines whether an target abort occurred.	

prop (CLI Abbreviation)	Property Description		Clearing the Status
	val	Value Description	
BX_STAT_RESETCODE (resetcode)	3-bit value	The state of DEVSEL# (bit 0), STOP# (bit 1) and TRDY# (bit 2) during the rising edge of RST#.	A clear has no effect.
BX_STAT_PIGGYBACKID (piggybackid)	Returns the ID of the piggyback board (2-bit value).		
	0	No piggyback board present.	
	1	Analyzer Piggyback Board (Option #100) detected.	
	2	Logic Analyzer Connection Board (Option #120) detected.	
BX_STAT_TRIGIO (trigio)	4-bit value	The current state of the trigger I/O pins.	
BX_STAT_CLK (clk)	Returns the state of the HW dip switch, which selects the operating frequency range. This makes it possible to use the card in a HW-emulation environment.		
	0	The testcard can only be run at specified PCI/PCI-X frequencies.	
	1	The testcard can be run in a HW-emulation environment. No PLL is used. The maximum frequency in this mode is 37 MHz.	
BX_STAT_PCIXMODE (pcixmode)	Indicates whether the bus runs in PCI or in PCI-X mode.		A clear has no effect.
	0	PCI mode.	
	1	PCI-X mode.	
BX_STAT_INVISIBLE (invisible)	0, 1	Returns the setting of the 'invisible' hardware jumper on the board. If set to '1', the configuration space cannot be accessed from the system under test. The testcard is invisible to the system configuration SW.	
BX_STAT_BUSSPEED (busspeed)	32-bit value	Returns the speed of the PCI/PCI-X Bus in Hertz. The accuracy is above 99 %.	
BX_STAT_BUSWIDTH (buswidth)	32-bit value	Width of the PCI/PCI-X Bus in bits. Either 32 or 64.	
BX_STAT_PCIRESET (pcireset)	1-bit value	Returns the current status of the PCI/PCI-X Bus Reset signal.	
BX_STAT_LASTRESET (lastreset)	32-bit value	Returns whether the PCI-X Bus has been reset since the last check.	BestXStatusClear with property BX_STAT_LASTRESET.

bx_versiontype

prop	CLI Abbreviation	Meaning
BX_VERSION_CAPI	capi	C-API version (software version number).
BX_VERSION_FIRMWARE	firmware	Firmware code version.
BX_VERSION_CORE	core	Core code version.
BX_VERSION_MEPHISTO	mephisto	Mephisto version.
BX_VERSION_TEAM	team	Team members of the PCI-X team.
BX_VERSION_BOARDID	boardid	Board ID number.
BX_VERSION_SERIALNO	serialno	Serial number of the main board.
BX_VERSION_ALTERA	altera	Version number of the Altera on the deeptrace board.
BX_VERSION_PRODUCT	product	Testcard type, either E2929A, E2929B, E2922A or E2922B.

bxppr_behpermtype

permprop (CLI Abbreviation)	Property Description	
	value/pValue	Value Description
BXPPR_BEHPERM_FIRSTPERM (firstperm)	Behavior first permutation. Starts the permutation algorithm at the specified point. If not all desired permutations have been exercised so far, you can continue the permutation by setting this number to where the last iteration ended. The last iteration can be queried with the property BXPPR_BEHRES_LASTPERM (see *"bxppr_behresulttype" on page 202*).	
	1 … MaxDWord default: 1	
BXPPR_BEHPERM_TUPLES (tuples)	Maximum number of groups that are permutated against each other for calculation of coverage.	

bxppr_behresulttype

resultprop (CLI Abbreviation)	value/pValue	Description
BXPPR_BEHRES_LASTPERM (lastperm)	0 … 2^{32}	Returns the last behavior permutation. This is the number of the last permutation of behaviors in the allocated behavior memory. This number can be used to determine which permutations will be covered after complete execution of the behavior memory.
BXPPR_BEHRES_TUPLES_LASTPERM (tupleslastperm)	0 … 2^{32}	Returns the last behavior permutation that is required to cover all n-tuples (see BXPPR_BEHPERM_TUPLES).
The following properties are only valid for requester-initiator behavior permutation:		
BXPPR_BEHRES_DATA (data)	0 … 2^{32}	Maximum amount of data in bytes that must be transferred to achieve complete coverage of behavior variations. This depends on the bus width.
BXPPR_BEHRES_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. This depends on the bus width.
BXPPR_BEHRES_TIME (time)	0 … 2^{32}	Returns the estimated test time in μs necessary to go through the complete behavior memory. This does not include the initial programming time. This depends on the bus width and speed.
BXPPR_BEHRES_TUPLES_TIME (tuplestime)	0 … 2^{32}	Returns the estimated time in μs required to transfer enough data to achieve complete coverage of all required group tuples.
The following properties are only valid for requester-initiator behavior permutation and only if requester-initiator block generation is enabled:		
BXPPR_BEHRES_RUNS (runs)	0 … 2^{32}	This property is valid only for requester-initiator behaviors. If block variations are set to be permutated, this returns the number of block memory runs that are needed for the complete coverage.
BXPPR_BEHRES_TUPLES_RUNS (tuplesruns)	0 … 2^{32}	If block variations are set to be permutated, this returns the number of block memory runs that are needed for group tuple coverage.

bxppr_gentype

genprop (CLI Abbreviation)	Property Description	
	value/pValue (CLI Abbreviation)	Value Description
BXPPR_GEN_USE_RIBLK (useriblk)	Defines whether to use requester-initiator block permutation.	
	default: BX_YES (yes), BX_NO (no)	
BXPPR_GEN_USE_RIBEH (useribeh)	Defines whether to use requester-initiator permutation.	
	default: BX_YES (yes), BX_NO (no)	
BXPPR_GEN_USE_RTBEH (usertbeh)	Defines whether to use requester-target permutation.	
	default: BX_YES (yes), BX_NO (no)	
BXPPR_GEN_USE_CTBEH (usectbeh)	Defines whether to use completer-target permutation.	
	default: BX_YES (yes), BX_NO (no)	
BXPPR_GEN_USE_CIBEH (usecibeh)	Defines whether to use completer-initiator permutation.	
	default: BX_YES (yes), BX_NO (no)	
BXPPR_GEN_ALGORITHM (alg)	Defines the algorithm used to permutate blocks and behaviors.	
	default: BXPPR_GEN_ALGORITHM_PERM (pprgenalgperm)	The "Classic" permutation algorithm. All values are selected one after the other as listed in the value list of the referring function.
	BXPPR_GEN_ALGORITHM_RANDOM (pprgenalgrand)	Random selection of values from the value list.
BXPPR_GEN_PRESET (preset)	Defines preset settings for PPR.	
	BXPPR_GEN_PRESET_DEFAULT	No special preset values. By default, each permutation list by default contains exactly one value, which is the default, and can be set to any parameter list. This mode works as the PCI version of PPR.
	BXPPR_GEN_PRESET_FIX	Each parameter list can be set only to a single value (the first one in any list), no permutation is generated.
	BXPPR_GEN_PRESET_FULL	No explicit parameter lists can be specified, all permutation lists are generated internally to guarantee that **every** possible parameter variation is generated.
BXPPR_GEN_BUSSPEED (busspeed)	PCI-X bus speed in Hz. Used to calculate times from clock cycles.	
	DWord value (0 – 0xffffffff) default: 66,000,000	

genprop (CLI Abbreviation)	Property Description	
	value/pValue (CLI Abbreviation)	**Value Description**
BXPPR_GEN_BUSWIDTH (buswidth)	PCI-X bus width of the used system in bits.	
	32, 64 default: 64	
BXPPR_GEN_SEED (seed)	Start value for the generation of pseudo random number sequences used by the permutations algorithm. This is used to make ”random” numbers reproducible.	
	DWord value (0 … 0xffffffff) default: 0	
BXPPR_GEN_XFERCLKS (xferclks)	Expected number of clocks per data transfer. This is used to estimate test times.	
	DWord value (0 … 0xffffffff) default: 0	
BXPPR_GEN_ADBLIMITATION (adblimitation)	If set to BX_YES, it prevents blocks from being created that can cause Mephisto bugs ADB_1, ADB_2 or ADB_3.	
	Mephisto 3.0: bx_yes … bx_yes default: bx_yes **Mephisto > 3.0**: bx_no … bx_yes default: bx_no	

bxppr_listtype

This type is simply a type definition to an array of characters.

```
value_list   :=    <list_entry> [, value_list]
list_entry   :=    [<repcount> *] [<value>]
value        :=    number (decimal format, hex format (xxx\h) or
                   binary format (01\b)
repcount     :=     number
```

bxppr_reporttype

reportprop (CLI Abbreviation)	**Property Description**	
	value/pValue	**Value Description**
BXPPR_REPORT_CAPI (capi)	Includes C-API abbreviations into the report.	
	BX_NO, BX_YES default: BX_NO	
BXPPR_REPORT_CONTENTS (contents)	Defines the length of the contents of the reports.	
	BXPPR_REPORT_CONTENTS_NO	No contents are printed.
	1-MaxDWord default: MaxDWord	Length of the contents in lines.

bxppr_riblkgaptype

gapprop (CLI Abbreviation)	Property Description	
	value/pValue (CLI Abbreviation)	**Value Description**
BXPPR_RIBLKGAP_BUSADDR_LO (busaddrlo)	Address of the gap.	
	0 … MaxDWord	
BXPPR_RIBLKGAP_BUSADDR_HI (busaddrhi)	High (upper 64 bit) part of gap address.	
	0 … MaxDWord	
BXPPR_RIBLKGAP_BUSCMD (buscmd)	Bus command that is needed to fill the gap (depends on the direction).	
	BX_RIBLK_BUSCMD_IO_READ (ioread)	I/O Read.
	BX_RIBLK_BUSCMD_IO_WRITE (iowrite)	I/O Write.
	BX_RIBLK_BUSCMD_MEM_READ_DWORD (memreaddword)	Memory Read DWORD.
	BX_RIBLK_BUSCMD_MEM_WRITE (memwrite)	Memory Write.
	BX_RIBLK_BUSCMD_ALIAS_MEM_READBLOCK (aliasmemreadblock)	For programming a PCI-X-compliant behavior, do **not** use this command. It will automatically be replaced by memreadblock.
	BX_RIBLK_BUSCMD_ALIAS_MEM_WRITEBLOCK (aliasmemwriteblock)	For programming a PCI-X-compliant behavior, do **not** use this command. It will automatically be replaced by memwriteblock.
	BX_RIBLK_BUSCMD_MEM_READBLOCK (memreadblock)	Memory Read Block.
	BX_RIBLK_BUSCMD_MEM_WRITEBLOCK (memwriteblock)	Memory Write Block.

gapprop (CLI Abbreviation)	**Property Description**	
	value/pValue (CLI Abbreviation)	**Value Description**
BXPPR_RIBLKGAP_BYTEN (byten)	Number of byte enables that are needed to fill this gap. The parameter is valid only for block transfers with command memwrite. Several useful constants have been predefined (same as in bxppr_riblktype).	
	0000/b … 1111/b	Refer to BX_RIBLK_BYTEN in *"bx_riblktype" on page 184*.
	BX_RIBLK_BYTEN_ALL (all)	Equal to byten = 0000/b.
	BX_RIBLK_BYTEN_DWORD0 (dword0)	Equal to byten = 0000/b.
	BX_RIBLK_BYTEN_NONE (none)	Equal to byten = 1111/b.
	BX_RIBLK_BYTEN_BYTE0 (byte0)	Equal to byten = 1110/b.
	BX_RIBLK_BYTEN_BYTE1 (byte1)	Equal to byten = 1101/b.
	BX_RIBLK_BYTEN_ BYTE2 (byte2)	Equal to byten = 1011/b.
	BX_RIBLK_BYTEN_ BYTE3 (byte3)	Equal to byten = 0111/b.
	BX_RIBLK_BYTEN_WORD0 (word0)	Equal to byten = 1100/b.
	BX_RIBLK_BYTEN_WORD1 (word1)	Equal to byten = 0011/b.
BXPPR_RIBLKGAP_NUMBYTES (numbytes)	Number of bytes that are needed to fill the gap for the current block.	
	0 … MaxDWord	

bxppr_riblkpermtype

blockprop (CLI Abbreviation)	Property Description	
	value/pValue (CLI Abbreviation)	Value Description
BXPPR_RIBLKPERM_DIRECTION (direction)	Direction of block transfer.	
	default: BXPPR_RIBLKPERM_DIRECTION_READ	Data transfer from system memory to exerciser.
	BXPPR_RIBLKPERM_DIRECTION_WRITE	Data transfer from exerciser to system memory.
	BXPPR_RIBLKPERM_DIRECTION_ READCOMPARE	The contents of system memory are compared with the exerciser contents.
	BXPPR_RIBLKPERM_DIRECTION_ WRITEREADCOMPARE	Data transfer from exerciser to system memory, read back and compare with original values.
BXPPR_RIBLKPERM_BLOCKSIZE (blocksize)	Size of the compound block that is transferred.	
	0 … MaxDWord default: 0	
BXPPR_RIBLKPERM_BUSADDRESS_LO (busaddrlo)	32-bit value	Start bus address of the compound block transfer (lower 32 bits).
BXPPR_RIBLKPERM_BUSADDRESS_HI (busaddrhi)	32-bit value	Start bus address of the compound block transfer (upper 32 bits).
BXPPR_RIBLKPERM_INTADDR (intaddr)	Offset of the internal resource (data memory or data generator) that is used.	
	0x00000 … 0x7ffff	See BX_RIBLK_INTADDR.
BXPPR_RIBLKPERM_RESOURCE (resource)	Internal Resource (data memory or data generator) that is used.	
	BXPPR_RIBLKPERM_RESOURCE_DATAMEM (datamem)	Data memory.
	BXPPR_RIBLKPERM_RESOURCE_DATAGEN (datagen)	Data generator. The type of the data generator can be determined with the generic exerciser property BX_EGEN_DATAGEN.
BXPPR_RIBLKPERM_FILLGAPS (fillgaps)	Defines whether gaps should be filled with fill blocks.	
	BX_NO, BX_YES default: BX_NO	
BXPPR_RIBLKPERM_FIRSTPERM (firstperm)	Behavior first permutation. Starts the permutation algorithm at the specified point. If not all desired permutations have been exercised so far, you can continue the permutation by setting this number to where the last iteration ended. The last iteration can be queried from property BXPPR_RIBLKRES_LASTPERM.	
	1 … MaxDWord default: 1	

blockprop (CLI Abbreviation)	Property Description	
	value/pValue (CLI Abbreviation)	Value Description
BXPPR_RIBLKPERM_STARTOFFSET (startoffset)	Start offset (see BestXRIBlkSet) for memory programming.	
	0 … 255 default: 0	Offset at memory line.
BXPPR_RIBLKPERM_SIZELIMIT (sizelimit)	Limits the requester-initiator block permutation memory usage.	
	default: BXPPR_RIBLKPERM_SIZELIMIT_NOLIMIT	No limitation on size. You can use the complete memory.
	1 … 255	Size limit in lines.
BXPPR_RIBLKPERM_TUPLES (tuples)	Maximum number of groups that are permutated against each other for a calculation of coverage.	

bxppr_riblkresulttype

blkprop (CLI Abbreviation)	value/pValue	Description
BXPPR_RIBLKRES_LASTPERM (lastperm)	0 … 2^{32}	Returns the last requester-initiator block permutation. This is the number of the last permutation of requester-initiator block results in the allocated block memory. This number can be used to determine which permutations will be covered after the requester-initiator block memory has been completely stepped through.
BXPPR_RIBLKRES_ACTUALSIZE (actualsize)	0 … 255	Returns the block memory size that is actually used. This size is always equal or below the value specified by the permutation property sizelimit (BXPPR_RIBLKPERM_SIZELIMIT).
BXPPR_RIBLKRES_NUMGAPS (numgaps)	0 … 2^{32}	Number of gaps left by the current block permutation.

bxppr_riblktype

blkprop (CLI Abbreviation)	Property Description	
	value/pValue (CLI Abbreviation)	Value Description
BXPPR_RIBLK_BUSCMD (buscmd)	Defines the bus commands that are used for the transfer. The constants are identical to the ones in *"bx_riblktype" on page 184*. I/O and Memory commands cannot be mixed.	
	BX_RIBLK_BUSCMD_IO_READ	I/O Read.
	BX_RIBLK_BUSCMD_IO_WRITE	I/O Write.
	BX_RIBLK_BUSCMD_MEM_READDWORD	Memory Read DWORD.
	BX_RIBLK_BUSCMD_MEM_WRITE	Memory Write.
	BX_RIBLK_BUSCMD_ALIAS_MEM_READBLOCK	For programming a PCI-X-compliant behavior, do **not** use this command. It will automatically be replaced by memreadblock.
	BX_RIBLK_BUSCMD_ALIAS_MEM_WRITEBLOCK	For programming a PCI-X-compliant behavior, do **not** use this command. It will automatically be replaced by memwriteblock.
	BX_RIBLK_BUSCMD_MEM_READBLOCK	Memory Read Block.
	BX_RIBLK_BUSCMD_MEM_WRITEBLOCK	Memory Write Block.
BXPPR_RIBLK_BYTEN (byten)	Byte enables that are used. This parameter is only valid for block transfers with the command memwrite.	
	0000/b – 1111/b	Refer to BX_RIBLK_BYTEN in *"bx_riblktype" on page 184*.
	BX_RIBLK_BYTEN_ALL	equal to byten = 0000/b.
	BX_RIBLK_BYTEN_DWORD0	Equal to byten = 0000/b.
	BX_RIBLK_BYTEN_NONE	Equal to byten = 1111/b.
	BX_RIBLK_BYTEN_BYTE0	Equal to byten = 1110/b.
	BX_RIBLK_BYTEN_BYTE1	Equal to byten = 1101/b.
	BX_RIBLK_BYTEN_BYTE2	Equal to byten = 1011/b.
	BX_RIBLK_BYTEN_BYTE3	Equal to byten = 0111/b.
	BX_RIBLK_BYTEN_WORD0	Equal to byten = 1100/b.
	BX_RIBLK_BYTEN_WORD1	Equal to byten = 0011/b.
BXPPR_RIBLK_ALIGN (align)	Alignment of the current block.	
	0 … 7 default: 0	Alignment to QWORD boundary.

	Property Description	
blkprop (CLI Abbreviation)	**value/pValue (CLI Abbreviation)**	**Value Description**
BXPPR_RIBLK_NUMBYTES (numbytes)	Number of bytes for the current block.	
	0 … MaxDWord	
BXPPR_RIBLK_RELAXORDER (relaxorder)	Relaxed ordering bit. It will be shown in the attribute phase.	
	0 , 1 default: 0	
BXPPR_RIBLK_NOSNOOP (nosnoop)	Bit that signifies that no snoop will be done. It will be shown in the attribute phase.	
	0, 1 default: 0	

Index

Publication Number: 5988-4898EN

Agilent Technologies