

Agilent E2929/E2930 Opt. 320 C-API/PPR

## **Programming Reference**



**Agilent Technologies**

## Important Notice

All information in this document is valid for both Agilent E2929 and Agilent E2930 testcards unless otherwise noted.

© Agilent Technologies, Inc. 2004

### Revision

April 2004

Printed in Germany

Agilent Technologies  
Herrenberger Straße 130  
D-71034 Böblingen  
Germany

Authors: t3 medien GmbH

### Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

### Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

### Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

### Safety Notices

#### CAUTION

A CAUTION notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a CAUTION notice until the indicated conditions are fully understood and met.

#### WARNING/DANGER

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

### Trademarks

Windows NT ® and MS Windows ® are U.S. registered trademarks of Microsoft Corporation.

# Contents

Documentation Overview	11
Introduction	13
Differences between the PCI and the PCI-X C-API/PPR Programming Interfaces	14
Naming Conventions	16
Structure of the PCI-X C-API/PPR Reference	17
Generic Functions	19
Initialization and Connection Functions	20
BestXClose	20
BestXDevIdentifierGet	21
BestXOpen	23
BestXPing	24
BestXRS232BaudRateSet	24
Administration Functions	26
BestXAllResourceUnlock	26
BestXCapabilityRead	27
BestXResourcesLocked	28
BestXResourceLock	29
BestXResourceUnlock	30
BestXVersionRead	31
Card Status Functions	32
BestXStatusClear	32
BestXStatusRead	33
Generic Control and Setup Functions	34
BestXAllDefaultSet	35
BestXBoardDefaultSet	35
BestXBoardGet	36
BestXBoardProg	37

BestXBoardRead	38
BestXBoardReset	38
BestXBoardSet	39
BestXPUProg	39
Power Management Event Functions	40
BestXPMERead	40
BestXPMEMWrite	41
Interrupt Generation Function	42
BestXInterruptGenerate	42
<b>Standard Analysis Functions</b>	<b>43</b>
Protocol Observer Functions	43
Compatibility Notes	44
BestXObsRead	45
BestXObsProg	46
BestXObsAccuErrorGet	47
BestXObsFirstErrorGet	48
BestXObsStatusGet	49
BestXObsRuleStringGet	50
BestXObsMaskSet	51
BestXObsMaskGet	52
BestXObsRuleGet	53
BestXObsBitPosGet	54
<b>Exerciser Functions</b>	<b>55</b>
Generic Exerciser Functions	56
BestXExerciserBreak	57
BestXExerciserContinue	57
BestXExerciserDefaultSet	58
BestXExerciserGenDefaultSet	58
BestXExerciserGenGet	59
BestXExerciserGenSet	60
BestXExerciserPause	60
BestXExerciserProg	61
BestXExerciserRead	61
BestXExerciserReset	62
BestXExerciserRun	63
BestXExerciserStop	63

<b>Requester-Initiator Programming Functions</b>	<b>64</b>
BestXRIBehDefaultSet	65
BestXRIBehGet	66
BestXRIBehMemInit	67
BestXRIBehSet	68
BestXRIBlockDefaultSet	69
BestXRIBlockGet	70
BestXRIBlockMemInit	70
BestXRIBlockSet	71
BestXRIGenDefaultSet	72
BestXRIGenGet	72
BestXRIGenSet	73
<b>Requester-Target Programming Functions</b>	<b>74</b>
BestXRTBehDefaultSet	75
BestXRTBehGet	76
BestXRTBehMemInit	76
BestXRTBehSet	77
BestXRTGenDefaultSet	78
BestXRTGenGet	78
BestXRTGenSet	79
<b>Completer-Target Programming Functions</b>	<b>80</b>
BestXCTBehDefaultSet	81
BestXCTBehGet	82
BestXCTBehMemInit	83
BestXCTBehSet	84
BestXCTGenDefaultSet	85
BestXCTGenGet	85
BestXCTGenSet	86
<b>Completer-Initiator Programming Functions</b>	<b>87</b>
BestXCIBehDefaultSet	88
BestXCIBehGet	89
BestXCIBehMemInit	90
BestXCIBehSet	91
BestXCIGenDefaultSet	92
BestXCIGenGet	92
BestXCIGenSet	93
<b>Configuration Space Programming Functions</b>	<b>94</b>
BestXConfRegGet	94

BestXConfRegMaskGet	95
BestXConfRegMaskSet	96
BestXConfRegSet	97
<b>Decoder Programming Functions</b>	<b>98</b>
BestXTDecoderAllSet	99
BestXTDecoderDefaultSet	100
BestXTDecoderGet	101
BestXTDecoderSet	102
BestXCTSplitCondDefaultSet	103
BestXCTSplitCondGet	104
BestXCTSplitCondSet	105
<b>Expansion ROM Programming Functions</b>	<b>106</b>
BestXExpRomRead	106
BestXExpRomWrite	107
<b>Data Memory Functions</b>	<b>108</b>
BestXDataMemInit	108
BestXDataMemRead	109
BestXDataMemWrite	110
<b>Host Access Functions</b>	<b>111</b>
BestXHostPCIRegRead	112
BestXHostPCIRegWrite	113
<b>Analyzer Functions</b>	<b>115</b>
<b>Pattern Term Function</b>	<b>115</b>
BestXPattProg	116
<b>Trigger Sequencer Programming Functions</b>	<b>121</b>
BestXTrigCondSet	122
BestXTrigDefaultSet	125
BestXTrigGenDefaultSet	125
BestXTrigGenGet	126
BestXTrigGenSet	127
BestXTrigProg	128
BestXTrigTranCondDefaultSet	129
BestXTrigTranSet	130
<b>Trace Memory Functions</b>	<b>131</b>
BestXTraceBitPosGet	132
BestXTraceBytePerLineGet	133

BestXTraceDataRead	134
BestXTraceDefaultWrite	135
BestXTraceDump	136
BestXTraceRead	137
BestXTraceRun	137
BestXTraceStop	138
BestXTraceWrite	139
<b>Performance Sequencer Programming Functions</b>	<b>140</b>
BestXPerfCondSet	141
BestXPerfCtrRead	142
BestXPerfDefaultSet	143
BestXPerfGenDefaultSet	144
BestXPerfGenGet	145
BestXPerfGenSet	146
BestXPerfProg	147
BestXPerfRun	148
BestXPerfTranCondDefaultSet	148
BestXPerfTranSet	149
BestXPerfStop	150
BestXPerfUpdate	150
<b>Interface Control Functions</b>	<b>151</b>
<b>Display Functions</b>	<b>151</b>
BestXDisplayStringWrite	152
BestXDisplayWrite	153
<b>Mailbox Functions</b>	<b>154</b>
BestXMailboxReceiveRegRead	154
BestXMailboxSendRegWrite	155
BestXPCICfgMailboxReceiveRegRead	156
BestXPCICfgMailboxSendRegWrite	157
<b>Protocol Permutator and Randomizer Functions</b>	<b>159</b>
<b>PPR Administration Functions</b>	<b>160</b>
BestXPprDelete	160
BestXPprInit	161
BestXPprProg	161
<b>PPR Generic Functions</b>	<b>162</b>
BestXPprGenDefaultSet	162

BestXPprGenGet	163
BestXPprGenSet	164
<b>PPR Requester-Initiator Functions</b>	<b>165</b>
BestXPprRIBehListDefaultSet	166
BestXPprRIBehListGet	167
BestXPprRIBehListSet	168
BestXPprRIBehPermDefaultSet	169
BestXPprRIBehPermGet	169
BestXPprRIBehPermSet	170
BestXPprRIBehResultGet	171
BestXPprRIBlkListDefaultSet	172
BestXPprRIBlkListGet	173
BestXPprRIBlkListSet	174
BestXPprRIBlkPermDefaultSet	175
BestXPprRIBlkPermGet	175
BestXPprRIBlkPermSet	176
BestXPprRIBlkResultGet	177
BestXPprRIDefaultSet	177
BestXPprRIBlkGapGet	178
<b>PPR Requester-Target Functions</b>	<b>179</b>
BestXPprRTBehListDefaultSet	180
BestXPprRTBehListGet	181
BestXPprRTBehListSet	182
BestXPprRTBehPermDefaultSet	183
BestXPprRTBehPermGet	184
BestXPprRTBehPermSet	185
BestXPprRTBehResultGet	186
BestXPprRTDefaultSet	186
<b>PPR Completer-Target Functions</b>	<b>187</b>
BestXPprCTBehListDefaultSet	188
BestXPprCTBehListGet	189
BestXPprCTBehListSet	190
BestXPprCTBehPermDefaultSet	191
BestXPprCTBehPermGet	191
BestXPprCTBehPermSet	192
BestXPprCTBehResultGet	193
BestXPprCTDefaultSet	193
<b>PPR Completer-Initiator Functions</b>	<b>194</b>



BestXPprCIBehListDefaultSet	195
BestXPprCIBehListGet	196
BestXPprCIBehListSet	197
BestXPprCIBehPermDefaultSet	198
BestXPprCIBehPermGet	198
BestXPprCIBehPermSet	199
BestXPprCIBehResultGet	200
BestXPprCIDefaultSet	200
PPR Reporting Functions	201
BestXPprReportWrite	201
BestXPprReportFile	202
BestXPprReportGet	202
BestXPprReportSet	203
Error Handling	205
Error Functions	205
BestXErrorStringGet	206
BestXLastErrorGet	207
BestXLastErrorStringGet	207
Type Definitions	209
bx_addrspacetype	209
bx_boardtype	210
bx_cibehtype	213
bx_cigentype	215
bx_ctbehtype	215
bx_ctgentype	217
bx_ctsplittype	218
bx_decproptype	220
bx_dectype	221
bx_egentype	222
bx_errtype	230
bx_obsruletype	231
bx_patttype	235
bx_perfcondtype	235

bx_perfgentype	236
bx_perftrantype	236
bx_porttype	237
bx_resourcetype	238
bx_ribehtype	239
bx_riblkttype	241
bx_rigentype	244
bx_rtbehtype	245
bx_rtgentype	246
bx_signaltype	247
Values of List Signals	250
bx_sizetype	252
bx_statustype	253
bx_tracetype	258
bx_trigcondtype	259
bx_triggentype	259
bx_trigtrantype	260
bx_versiontype	260
bxppr_behpermttype	261
bxppr_behresulttype	262
bxppr_gentype	263
bxppr_listtype	264
bxppr_reporttype	265
bxppr_riblkgaptype	266
bxppr_riblkpermttype	268
bxppr_riblkresulttype	269
bxppr_riblkttype	270

# Documentation Overview

This section shows you the different types of documents offered by Agilent Technologies and gives you an overview of which documents are available when you work with the E2929/E2930 PCI-X Exerciser and Analyzer.

All information is valid for both Agilent E2929 and Agilent E2930 testcards unless otherwise noted. The following documents are available:

## Getting Started Guide

- **Getting Started Guide**

Introduces standard analysis features and provides an example of how to set up the protocol observer.

Provides an overview of how the Agilent E2930 testcard supports the PCI-X 2.0 features.

Gives detailed information about hardware and interfaces.

## User's Guides

- **E2929/E2930 Opt. 300 PCI-X Exerciser User's Guide**

Provides information on programming the testcard as an initiator and/or target device. It shows you how to actively stimulate the PCI-X bus.

This guide shows how to:

- Initiate data transfers on the PCI-X bus (act as requester-initiator).
- Act as completer-target.
- Handle split-completion transactions (act as completer-initiator).
- Handle open requests (act as requester-target).

- **E2929/E2930 PCI-X Analyzer User's Guide**

Provides information on how to examine the behavior and performance of a PCI-X device on the bus and shows how to perform functional tests such as data compares.

- **E2929 Opt. 200 PCI-X Performance Optimizer User's Guide**

Provides all features that are needed to evaluate and optimize any device under test in terms of the performance (post-processed performance analysis and optimization). Option 200 is available for Agilent E2929 testcards only.

- **Agilent E2920 PCI-X Series Opt. 320 C-API/PPR Programmer's Guide**

Provides information on how to set up test programs using the C functions described in the corresponding C-API/PPR Reference.

**GUI and C-API/PPR References**

- **E2929/E2930 Windows and Dialog Boxes Reference**

Provides reference information on all windows and dialog boxes of the Agilent E2920 graphical user interface (GUI).

- **E2929/E2930 Opt. 320 C-API/PPR Reference**

Describes all C functions, types and definitions of the application programming interface of the E2929/E2930 PCI-X testcard.

This reference also provides the commands and abbreviations that are used in the command line interface (CLI) of the graphical user interface.

- **Agilent E2922/E2923 Opt. 320 C-API/PPR Reference**

Describes all C functions, types and definitions of the application programming interface of the Agilent E2922/E2923 PCI-X testcard.

This reference also provides the commands and abbreviations that are used in the command line interface (CLI) of the graphical user interface.

# 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 14* 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 15* introduces two ways of programming the PCI-X testcard. These are C-API and CLI.
- *“Naming Conventions” on page 16* 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 17* 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, the exerciser status, the observer status and the trace memory status.

## Changes in Terminology for the Exerciser

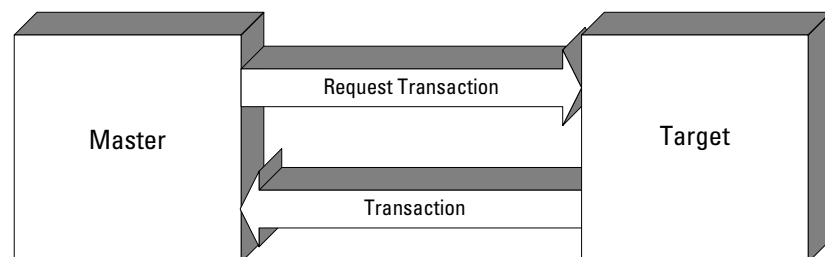
The PCI-X Specification introduces the following changes in terminology:

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

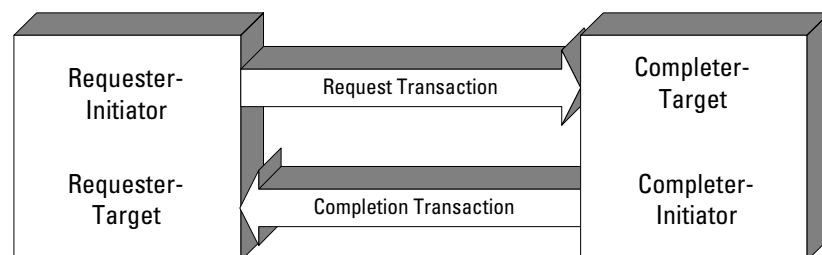
The exerciser now consists of initiator and target.

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.
Analyzer Functions	These functions are used for trace memory controlling, performance measurements and protocol checking.
Interface Control Functions	These functions are used to control the 7-segment display and the status of the power management event pin.
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
<i>"BestXDevIdentifierGet" on page 21</i>	Returns the identifier of a PCI-X device.
<i>"BestXOpen" on page 23</i>	Opens a connection to the testcard.
<i>"BestXClose" on page 20</i>	Closes the connection to the testcard.
<i>"BestXPing" on page 24</i>	Checks a connection to the testcard.
<i>"BestXRS232BaudRateSet" on page 24</i>	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 230*.

**Input Parameters** **handle** Session identification.

**See also** *"BestXOpen" on page 23*

## 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 22.

**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 230.

**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.

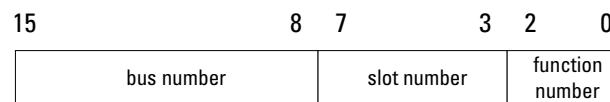
**Output Parameters** **devid** Device identifier within the system. This is the device number used to access the testcard’s configuration space (for example, in “*BestXOpen*” on page 23).

**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).

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

**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 230.

**Input Parameters** **port** Type of control port; see “*bx\_porttype*” on page 237.

**portnum** Control port; see “*bx\_porttype*” on page 237.

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

**See also** “*BestXClose*” on page 20  
 “*BestXDevIdentifierGet*” on page 21

## 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 230.

**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 230.

**Input Parameters** **handle** Session identification.

**baudrate** Baud rate value; see “*Baud Rate Values*” on page 25.

**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
<i>"BestXVersionRead" on page 31</i>	Checks the version of a component of the testcard.
<i>"BestXCapabilityRead" on page 27</i>	Reads the OR-combined capability code.
<i>"BestXResourceLock" on page 29</i>	Locks a resource.
<i>"BestXResourceIsLocked" on page 28</i>	Checks whether a resource is locked.
<i>"BestXResourceUnlock" on page 30</i>	Unlocks a resource.
<i>"BestXAllResourceUnlock" on page 26</i>	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 230*.

**Input Parameters** **handle** Session identification.

**See also** *"BestXResourceUnlock" on page 30*  
*"BestXResourceIsLocked" on page 28*  
*"BestXResourceLock" on page 29*

## 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 230.

**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_ANALYZER	The testcard can run the analyzer with the trace memory.
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 230.

**Input Parameters** **handle** Session identification.

**resource** Resource to be checked; see “*bx\_resourcetype*” on page 238.

**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 237. If `BX_PORT_CURRENT` is returned, the resource is locked by the current session.

**See also** “*BestXResourceUnlock*” on page 30  
 “*BestXAllResourceUnlock*” on page 26  
 “*BestXResourceLock*” on page 29  
 “*BestXPing*” on page 24

## 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 28.*

**CLI Equivalent**     `BestXResourceLock resource=<resource>`

**CLI Abbreviation**     `reslock res=<resource>`

**Return Value**     Error code; see *“bx\_errtype” on page 230.*

**handle**     Session identification.

**resource**     Resource to be locked; see *“bx\_resourcetype” on page 238.*

**See also**     *“BestXResourceUnlock” on page 30*  
                 *“BestXAllResourceUnlock” on page 26*

## 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 230.

**Input Parameters** **handle** Session identification.

**resource** Resource to be unlocked; see “*bx\_resourcetype*” on page 238.

**See also** “*BestXResourceIsLocked*” on page 28  
 “*BestXResourceLock*” on page 29  
 “*BestXAllResourceUnlock*” on page 26

## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Version property to be read; see “*bx\_versiontype*” on page 260.

**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 260

# Card Status Functions

The following functions are used to access status register contents:

Function	Action
<i>"BestXStatusRead" on page 33</i>	Reads the content of a specific status register.
<i>"BestXStatusClear" on page 32</i>	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 230*.

**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 253*.

**See also** *"BestXStatusRead" on page 33*



## 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
- trace memory status
- observer status

**CLI Equivalent** `BestXStatusRead prop=<prop>`

**CLI Abbreviation** `statusread prop=<prop>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**prop** Status register property; see “*bx\_statustype*” on page 253.

**Output Parameters** **val** Status register content; see “*bx\_statustype*” on page 253.

**See also** “*BestXStatusClear*” on page 32

# Generic Control and Setup Functions

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

Function	Action
<i>"BestXPUProg" on page 39</i>	Stores the current setting of the configuration space into the non-volatile memory.
<i>"BestXAllDefaultSet" on page 35</i>	Resets all properties on the controlling host to default values.
<i>"BestXBoardDefaultSet" on page 35</i>	Sets the generic testcard properties to default values.
<i>"BestXBoardSet" on page 39</i>	Sets a generic testcard property.
<i>"BestXBoardGet" on page 36</i>	Gets a generic testcard property.
<i>"BestXBoardReset" on page 38</i>	Resets the testcard.
<i>"BestXBoardRead" on page 38</i>	Reads testcard properties from the testcard.
<i>"BestXBoardProg" on page 37</i>	Writes testcard properties to the testcard.
<i>"BestXPMEWrite" on page 41</i>	Issues a power management event.
<i>"BestXPMERead" on page 40</i>	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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPUProg*” on page 39

## 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 210.

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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXBoardGet*” on page 36  
“*BestXBoardProg*” on page 37  
“*BestXBoardRead*” on page 38  
“*BestXBoardReset*” on page 38  
“*BestXBoardSet*” on page 39

## 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 230.

**Input Parameters**    **handle**    Session identification.

**prop**    Board property to be obtained; see “*bx\_boardtype*” on page 210.

**Output Parameters**    **val**    Value of the board property; see “*bx\_boardtype*” on page 210.

**See also**    “*BestXBoardDefaultSet*” on page 35  
                  “*BestXBoardProg*” on page 37  
                  “*BestXBoardRead*” on page 38  
                  “*BestXBoardReset*” on page 38  
                  “*BestXBoardSet*” on page 39

## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Board property; see “*bx\_boardtype*” on page 210.

**Output Parameters** **value** Value of the board property; see “*bx\_boardtype*” on page 210.

**See also** “*BestXBoardGet*” on page 36  
“*BestXBoardDefaultSet*” on page 35  
“*BestXBoardRead*” on page 38  
“*BestXBoardReset*” on page 38  
“*BestXBoardSet*” on page 39

## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Board property; see “*bx\_boardtype*” on page 210.

**value** Value of the board property; see “*bx\_boardtype*” on page 210.

**See also** “*BestXBoardGet*” on page 36  
 “*BestXBoardDefaultSet*” on page 35  
 “*BestXBoardProg*” on page 37  
 “*BestXBoardReset*” on page 38  
 “*BestXBoardSet*” on page 39

## 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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXBoardGet*” on page 36  
 “*BestXBoardDefaultSet*” on page 35  
 “*BestXBoardProg*” on page 37  
 “*BestXBoardRead*” on page 38  
 “*BestXBoardSet*” on page 39

## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Property to be set; see “*bx\_boardtype*” on page 210.

**val** Value to which the property is set; see “*bx\_boardtype*” on page 210.

**See also** “*BestXBoardGet*” on page 36  
           “*BestXBoardDefaultSet*” on page 35  
           “*BestXBoardProg*” on page 37  
           “*BestXBoardRead*” on page 38  
           “*BestXBoardReset*” on page 38

## 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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXAllDefaultSet*” on page 35

# Power Management Event Functions

The following functions are used to supervise power management events:

Function	Action
<i>"BestXPMEWrite" on page 41</i>	Defines the status of the power management enable pin.
<i>"BestXPMERead" on page 40</i>	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** pmeread

**Return Value** Error code; see *"bx\_errtype" on page 230*.

**Input Parameters** **handle** Session identification.

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

- 0 – no event
- 1 – event

**See also** *"BestXPMEWrite" on page 41*



## 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 230.

**Input Parameters** **handle** Session identification.

**value** Value of the pin:

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

**See also** “*BestXPMERead*” on page 40

# Interrupt Generation Function

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

Function	Action
<i>"BestXInterruptGenerate" on page 42</i>	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 32*.

**CLI Equivalent** `BestXInterruptGenerate intr=<intr>`

**CLI Abbreviation** `interruptgenerate intr=<intr>`

**Return Value** Error code; see *"bx\_errtype" on page 230*.

**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	inta	PCI-X Interrupt Request A ... D (INTA# ... INTD#)
...	...	
BX_INTD	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
<i>"BestXObsRead" on page 45</i>	Reads all mask and error values from the testcard to the host.
<i>"BestXObsProg" on page 46</i>	Updates mask values on the testcard.
<i>"BestXObsAccuErrorGet" on page 47</i>	Returns the accumulated error value for a specific rule.
<i>"BestXObsFirstErrorGet" on page 48</i>	Returns the first error value for a specific rule.
<i>"BestXObsStatusGet" on page 49</i>	Returns a textual error summary.
<i>"BestXObsRuleStringGet" on page 50</i>	Returns the protocol rule text for a specific error.
<i>"BestXObsMaskSet" on page 51</i>	Sets the mask value for a specific rule.
<i>"BestXObsMaskGet" on page 52</i>	Returns the mask value of a specific rule.
<i>"BestXObsRuleGet" on page 53</i>	Returns the protocol rule identifier for a specific error.
<i>"BestXObsBitPosGet" on page 54</i>	Gets the bit position of the specified observer rule.

## Compatibility Notes

The concept of the protocol observer interface has changed with the introduction of the E2930 testcard. This interface is also valid for E2929 testcards.

With this new interface, reading and writing protocol observer data is a two-step process:

- For reading, first the data is read as a whole from the testcard to the host. The individual information can then be retrieved from the data on the host.
- For writing, the mask data is first written to the host data and then programmed to the testcard.

To adapt existing programs to this new concept, consider the following:

- Replace `BestXObsMaskProg` by
  - one or more calls to `BestXObsMaskSet` and
  - a final call to `BestXObsProg`.
- Replace `BestXObsMaskRead` by
  - a call to `BestXObsRead` and
  - one or more calls to `BestXObsMaskGet`.
- Replace `BestXObsStatusRead` by
  - `BestXObsRead` and
  - `BestXObsStatusGet`.
- Replace calls to `BestXStatusRead` using parameters `BX_STAT_OBS_FIRST_LO`, `BX_STAT_OBS_FIRST_HI`, `BX_STAT_OBS_ACCU_LO` or `BX_STAT_OBS_ACCU_HI` by
  - `BestXObsRead` and
  - one or more calls to `BestXObsAccuErrorGet` or `BestXObsFirstErrorGet`.

## BestXObsRead

**Call** `bx_errtype BestXObsRead( bx_handletype handle );`

**Description** Reads all mask and error values from the testcard to the host. After this data has been read, the various "...Get" functions can be used to retrieve details about specific rules.

**CLI Equivalent** `BestXObsRead`

**CLI Abbreviation** `obsread`

**Return Value** Error code; see "*bx\_errtype*" on page 230.

**Input Parameters** **handle** Session identification.

**See also** "*BestXObsAccuErrorGet*" on page 47

"*BestXObsFirstErrorGet*" on page 48

"*BestXObsStatusGet*" on page 49

"*BestXObsRuleStringGet*" on page 50

"*BestXObsMaskGet*" on page 52

"*BestXObsRuleGet*" on page 53

"*BestXObsBitPosGet*" on page 54

## BestXObsProg

**Call** `bx_errtype BestXObsProg( bx_handletype handle );`

**Description** Copies all mask values from the host to the testcard and thus updates the values on the testcard.

To set mask values for specific rules, call `BestXObsMaskSet` beforehand.

**CLI Equivalent** `BestXObsProg`

**CLI Abbreviation** `obsprog`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXObsMaskSet*” on page 51

## BestXObsAccuErrorGet

**Call** `bx_errtype BestXObsAccuErrorGet (`  
     `bx_handletype handle`  
     `bx_obsruletype rule,`  
     `bx_int32 *accuerr );`

**Description** Returns the accumulated error value for a specific rule. The accumulated error indicates whether the rule was violated since the last error reset with `BestXStatusClear (BX_STAT_OBS_ERR)` or `BestXObsProg`.

This function does *not* access the testcard, but reads the values stored on the host. To update the error values on the host, call `BestXObsRead` beforehand.

**CLI Equivalent** `BestXObsAccuErrorGet rule=<rule>`

**CLI Abbreviation** `obsaccuerrorget rule=<rule>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**rule** Identifier of the observer rule; see “*bx\_obsruletype*” on page 231.

**Output Parameters** **accuerr** Accumulated error value for the selected rule (1 if the rule was violated, 0 otherwise).

**See also** “*BestXObsRead*” on page 45

“*BestXObsFirstErrorGet*” on page 48

“*BestXStatusClear*” on page 32

“*BestXObsProg*” on page 46

## BestXObsFirstErrorGet

**Call** `bx_errtype BestXObsFirstErrorGet(  
           bx_handletype    handle  
           bx_obsruletype   rule,  
           bx_int32         *firsterr );`

**Description** Returns the first error value for a specific rule. The first error value is 1 if the selected rule was the first rule violated after the latest error reset with `BestXStatusClear(BX_STAT_OBS_ERR)` or `BestXObsProg`.

This function does *not* access the testcard, but reads the values stored on the host. To update the error values on the host, call `BestXObsRead` beforehand.

**CLI Equivalent** `BestXObsFirstErrorGet rule=<rule>`

**CLI Abbreviation** `obsfirsterrorget rule=<rule>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**rule** Identifier of the observer rule; see “*bx\_obsruletype*” on page 231.

**Output Parameters** **firsterr** First error value for the selected rule (1 if the selected rule was the first rule violated after the latest reset, 0 otherwise).

**See also** “*BestXObsRead*” on page 45

“*BestXObsAccuErrorGet*” on page 47

“*BestXStatusClear*” on page 32

“*BestXObsProg*” on page 46



## BestXObsStatusGet

**Call** `bx_errtype BestXObsStatusGet(  
 bx_handletype handle  
 bx_charptrtype *statustext );`

**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.

This function does *not* access the testcard, but reads the values stored on the host. To update the values on the host, call `BestXObsRead` beforehand.

**CLI Equivalent** `BestXObsStatusGet`

**CLI Abbreviation** `obsstatusget`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**Output Parameters** **statustext** Error summary text.

**See also** “*BestXObsRead*” on page 45

## BestXObsRuleStringGet

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

**Description** Returns a text string that describes the specified rule (usually called after identifying a violated rule with `BestXObsAccuErrorGet` or `BestXObsFirstErrorGet`).

**CLI Equivalent** `BestXObsRuleStringGet rule=<rule>`

**CLI Abbreviation** `obsrulestringget rule=<rule>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**rule** Bit position that specifies the rule; see “*bx\_obsruletype*” on page 231.

**Output Parameters** **ruletext** Text string that describes the rule.

**See also** “*BestXObsRead*” on page 45  
“*BestXObsRuleGet*” on page 53

## BestXObsMaskSet

**Call**    `bx_errtype BestXObsMaskSet (  
          bx_handletype    handle  
          bx_obsruletype   rule,  
          bx_int32         mask );`

**Description**    Sets the mask value for a specific rule. This function does *not* access the testcard, but sets the values on the host.

To update the mask value on the testcard, call `BestXObsProg` afterwards.

**CLI Equivalent**    `BestXObsMaskSet rule=<rule> mask=<mask>`

**CLI Abbreviation**    `obsmaskset rule=<rule> mask=<mask>`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**    **handle**    Session identification.

**rule**    Identifier of the observer rule; see “*bx\_obsruletype*” on page 231.

**mask**    Mask value for the selected rule. Set to 1 to disable the rule, set to 0 to enable the rule.

**See also**    “*BestXObsProg*” on page 46

## BestXObsMaskGet

**Call** `bx_errtype BestXObsMaskGet (`  
    `bx_handletype handle`  
    `bx_obsruletype rule,`  
    `bx_int32 *mask );`

**Description** Returns the mask value of a specific rule. This function does *not* access the testcard, but reads the values stored on the host.

To update the mask values on the host, call `BestXObsRead` beforehand.

**CLI Equivalent** `BestXObsMaskGet rule=<rule>`

**CLI Abbreviation** `obsmaskget rule=<rule>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**rule** Identifier of the observer rule; see “*bx\_obsruletype*” on page 231.

**Output Parameters** **mask** Mask value of the selected rule (1 if the rule is disabled, 0 if the rule is enabled).

**See also** “*BestXObsRead*” on page 45

## 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.  
This function does *not* access the testcard.

**CLI Equivalent**   `BestXObsRuleGet bitpos=<bitpos>`

**CLI Abbreviation**   `obsruleget bitpos=<bitpos>`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**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 231.

**See also**        “*BestXObsRead*” on page 45

                  “*BestXObsBitPosGet*” on page 54

## BestXObsBitPosGet

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

**Description** Gets the bit position of the specified observer rule.  
This function does *not* access the testcard.

**CLI Equivalent** `BestXObsBitPosGet rule=<rule>`

**CLI Abbreviation** `obsbitposget rule=<rule>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**rule** Identifier of the observer rule; see “*bx\_obsruletype*” on page 231.

**Output Parameters** **bitpos** Bit position in one of the error registers that specifies the rule.

**See also** “*BestXObsRead*” on page 45  
“*BestXObsRuleGet*” on page 53

# 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
<i>"BestXExerciserDefaultSet" on page 58</i>	Sets all exerciser properties to default values.
<i>"BestXExerciserGenDefaultSet" on page 58</i>	Sets the exerciser generic properties to default values.
<i>"BestXExerciserGenSet" on page 60</i>	Sets an exerciser generic property.
<i>"BestXExerciserGenGet" on page 59</i>	Gets an exerciser generic property.
<i>"BestXExerciserProg" on page 61</i>	Loads exerciser settings and properties to the testcard.
<i>"BestXExerciserRun" on page 63</i>	Resets all initiator and target behavior counters and starts requester-initiator transactions.
<i>"BestXExerciserRead" on page 61</i>	Reads all Exerciser properties from the testcard to the host storage.
<i>"BestXExerciserStop" on page 63</i>	Stops the requester-initiator.
<i>"BestXExerciserBreak" on page 57</i>	Resets the complete exerciser and leaves the bus in an clear state.
<i>"BestXExerciserPause" on page 60</i>	Completes the current transaction and halts the requester-initiator.
<i>"BestXExerciserContinue" on page 57</i>	Runs requester-initiator transactions without resetting counters.
<i>"BestXExerciserReset" on page 62</i>	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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXExerciserReset*” on page 62

## 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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXExerciserReset*” on page 62

## 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 230.

**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 222.

**CLI Equivalent** `BestXExerciserGenDefaultSet`

**CLI Abbreviation** `egendefset`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXExerciserGenSet*” on page 60  
 “*BestXExerciserGenGet*” on page 59

## 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 230.

**Input Parameters**    **handle**    Session identification.

**prop**    Exerciser property to be obtained; see “*bx\_egentype*” on page 222.

**Output Parameters**    **val**    Value of the exerciser property; see “*bx\_egentype*” on page 222.

**See also**    “*BestXExerciserGenSet*” on page 60  
              “*BestXExerciserGenDefaultSet*” on page 58  
              “*BestXExerciserProg*” on page 61

## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Exerciser property to be set; see “*bx\_egentype*” on page 222.

**val** Value to which the exerciser property is set; see “*bx\_egentype*” on page 222.

**See also** “*BestXExerciserProg*” on page 61  
     “*BestXExerciserGenGet*” on page 59  
     “*BestXExerciserGenDefaultSet*” on page 58

## 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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXExerciserContinue*” on page 57

## 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 230.

**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 230.

**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 61

## 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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXExerciserBreak*” on page 57

## 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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXExerciserStop*” on page 63  
“*BestXExerciserContinue*” on page 57  
“*BestXExerciserPause*” on page 60  
“*BestXRIGenSet*” on page 73  
“*bx\_rigetype*” on page 244

## 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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXExerciserRun*” on page 63  
“*BestXExerciserContinue*” on page 57  
“*BestXExerciserPause*” on page 60

# 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
<i>"BestXRIGenDefaultSet" on page 72</i>	Sets all generic requester-initiator properties to default values.
<i>"BestXRIGenSet" on page 73</i>	Sets generic requester-initiator properties.
<i>"BestXRIGenGet" on page 72</i>	Gets generic requester-initiator properties.
<i>"BestXRIBlockMemInit" on page 70</i>	Initializes the requester-initiator block memory.
<i>"BestXRIBlockDefaultSet" on page 69</i>	Sets all requester-initiator block properties to default values.
<i>"BestXRIBlockSet" on page 71</i>	Sets requester-initiator block properties.
<i>"BestXRIBlockGet" on page 70</i>	Gets requester-initiator block properties.
<i>"BestXRIBehDefaultSet" on page 65</i>	Sets all requester-initiator behavior properties to default values.
<i>"BestXRIBehSet" on page 68</i>	Sets all requester-initiator behavior properties.
<i>"BestXRIBehGet" on page 66</i>	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\_ribetype*” on page 239.

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 230.

**Input Parameters**    **handle**    Session identification.

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

**See also**    “*BestXRIBehSet*” on page 68  
              “*BestXRIBehGet*” on page 66  
              “*BestXExerciserProg*” on page 61

## BestXRIBehGet

**Call**    `bx_errtype BestXRIBehGet (`  
           `bx_handletype    handle,`  
           `bx_int32            offset,`  
           `bx_ribetype        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 230.

**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\_ribetype*” on page 239.

**Output Parameters**    **val**    Value of the requester-initiator behavior property; see “*bx\_ribetype*” on page 239.

**See also**    “*BestXRIBehDefaultSet*” on page 65  
                   “*BestXRIBehSet*” on page 68  
                   “*BestXRIBlockGet*” on page 70

## 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 239.

**CLI Equivalent** `BestXRIBehMemInit`

**CLI Abbreviation** `ribehmeminit`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXRIBlockMemInit*” on page 70  
“*BestXCIBehMemInit*” on page 90  
“*BestXRTBehMemInit*” on page 76  
“*BestXCTBehMemInit*” on page 83

## BestXRIBehSet

**Call** `bx_errtype BestXRIBehSet (`  
     `bx_handletype handle,`  
     `bx_int32 offset,`  
     `bx_ribetype 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 230.

**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\_ribetype*” on page 239.

**val** Value of the requester-initiator behavior property to be set; see “*bx\_ribetype*” on page 239.

**See also** “*BestXRIBehDefaultSet*” on page 65  
     “*BestXRIBehGet*” on page 66  
     “*BestXRIBlockSet*” on page 71

## 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** `ribblkdefset offset=<offset>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

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

**See also** “*BestXRIBlockSet*” on page 71  
          “*BestXRIBlockGet*” on page 70  
          “*BestXRIBehDefaultSet*” on page 65

## 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** `ribblkget offset=<offset> prop=<prop>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**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 241.

**Output Parameters** **val** Value of the requester-initiator block property; see “*bx\_riblktype*” on page 241.

**See also** “*BestXRIBlockDefaultSet*” on page 69  
 “*BestXRIBlockSet*” on page 71  
 “*BestXRIBehGet*” on page 66

## 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 241.

**CLI Equivalent** `BestXRIBlockMemInit`

**CLI Abbreviation** `riblkmeminit`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXRIBehMemInit*” on page 67

## 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 230.

**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 241.

**val** Value of the requester-initiator block property to be set; see “*bx\_riblktype*” on page 241.

**See also** “*BestXRIBlockDefaultSet*” on page 69  
     “*BestXRIBlockGet*” on page 70  
     “*BestXRIBehSet*” on page 68  
     “*BestXExerciserProg*” on page 61

## 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 244.

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 230.

**Input Parameters** **handle** Session identification.

**See also** “BestXRIGenSet” on page 73  
“BestXRIGenGet” on page 72

## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Property to be obtained; see “*bx\_rigentype*” on page 244.

**Output Parameters** **val** Property value; see “*bx\_rigentype*” on page 244.

**See also** “BestXRIGenDefaultSet” on page 72  
“BestXRIGenSet” on page 73



## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Property to be set; see “*bx\_rigentype*” on page 244.

**val** Value to which the property is set; see “*bx\_rigentype*” on page 244.

**See also** “*BestXRIGenDefaultSet*” on page 72  
     “*BestXRIGenGet*” on page 72  
     “*BestXExerciserProg*” on page 61

# Requester-Target Programming Functions

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

Function	Action
<i>"BestXRTGenDefaultSet" on page 78</i>	Sets all generic requester-target properties to default values.
<i>"BestXRTGenSet" on page 79</i>	Sets generic requester-target properties.
<i>"BestXRTGenGet" on page 78</i>	Gets generic requester-target properties.
<i>"BestXRTBehMemInit" on page 76</i>	Initializes the requester-target behavior memory.
<i>"BestXRTBehDefaultSet" on page 75</i>	Sets all requester-target behavior properties on one line of the behavior memory to default values.
<i>"BestXRTBehSet" on page 77</i>	Sets a requester-target behavior property on one line of the behavior memory.
<i>"BestXRTBehGet" on page 76</i>	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 245.

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 230.

**Input Parameters**    **handle**    Session identification.

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

**See also**    “*BestXRTBehSet*” on page 77  
              “*BestXRTBehGet*” on page 76  
              “*BestXExerciserProg*” on page 61

## 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 230.

**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 245.

**Output Parameters** **val** Value of the requester-target behavior property; see “*bx\_rtbehtype*” on page 245.

**See also** “*BestXRTBehDefaultSet*” on page 75  
 “*BestXRTBehSet*” on page 77

## 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 245.

**CLI Equivalent** `BestXRTBehMemInit`

**CLI Abbreviation** `rtbehmeminit`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXRIBehMemInit*” on page 67  
 “*BestXCIBehMemInit*” on page 90  
 “*BestXCTBehMemInit*” on page 83

## 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 230.

**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 245.

**val** Value of the requester-target behavior property to be set; see “*bx\_rtbehtype*” on page 245.

**See also** “*BestXRTBehDefaultSet*” on page 75  
     “*BestXRTBehGet*” on page 76  
     “*BestXExerciserProg*” on page 61

## 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 246.

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

**CLI Equivalent** `BestXRTGenDefaultSet`

**CLI Abbreviation** `rtgndefset`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXRTGenSet*” on page 79  
 “*BestXRTGenGet*” on page 78  
 “*BestXExerciserProg*” on page 61

## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Property to be obtained; see “*bx\_rtgentype*” on page 246.

**Output Parameters** **val** Property value; see “*bx\_rtgentype*” on page 246.

**See also** “*BestXRTGenDefaultSet*” on page 78  
 “*BestXRTGenSet*” on page 79

## 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 230.

**Input Parameters**    **handle**    Session identification.

**prop**    Property to be set; see “*bx\_rtgentype*” on page 246.

**val**    Value to which the property is set; see “*bx\_rtgentype*” on page 246.

**See also**    “*BestXRTGenDefaultSet*” on page 78  
                   “*BestXRTGenGet*” on page 78  
                   “*BestXExerciserProg*” on page 61

# Completer-Target Programming Functions

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

Function	Action
<i>"BestXCTGenDefaultSet" on page 85</i>	Sets all generic completer-target properties to default values.
<i>"BestXCTGenSet" on page 86</i>	Sets generic completer-target properties.
<i>"BestXCTGenGet" on page 85</i>	Gets generic completer-target properties.
<i>"BestXCTBehMemInit" on page 83</i>	Initializes the completer-target behavior memory.
<i>"BestXCTBehDefaultSet" on page 81</i>	Sets all completer-target behavior properties on one line of the behavior memory to default values.
<i>"BestXCTBehSet" on page 84</i>	Sets a completer-target behavior property on one line of the behavior memory.
<i>"BestXCTBehGet" on page 82</i>	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 215.

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 230.

**Input Parameters**    **handle**    Session identification.

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

**See also**    “*BestXCTBehSet*” on page 84  
              “*BestXCTBehGet*” on page 82  
              “*BestXExerciserProg*” on page 61

## 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 230.

**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 215.

**Output Parameters** **val** Value of the completer-target behavior property; see “*bx\_ctbehtype*” on page 215.

**See also** “*BestXCTBehDefaultSet*” on page 81  
 “*BestXCTBehSet*” on page 84

## 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 215.

**CLI Equivalent** `BestXCTBehMemInit`

**CLI Abbreviation** `ctbehmeminit`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXRIBehMemInit*” on page 67  
“*BestXCIBehMemInit*” on page 90  
“*BestXRTBehMemInit*” on page 76

## 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 230.

**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 215.

**val**    Value of the completer-target behavior property to be set; see “*bx\_ctbehtype*” on page 215.

**See also**    “*BestXCTBehDefaultSet*” on page 81  
                  “*BestXCTBehGet*” on page 82  
                  “*BestXExerciserProg*” on page 61

## 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 217.

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

**CLI Equivalent** `BestXCTGenDefaultSet`

**CLI Abbreviation** `ctgndefset`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXCTGenSet*” on page 86  
 “*BestXCTGenGet*” on page 85  
 “*BestXExerciserProg*” on page 61

## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Property to be obtained; see “*bx\_ctgentype*” on page 217.

**Output Parameters** **val** Property value; see “*bx\_ctgentype*” on page 217.

**See also** “*BestXCTGenDefaultSet*” on page 85  
 “*BestXCTGenSet*” on page 86

## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Property to be set; see “*bx\_ctgentype*” on page 217.

**val** Value to which the property is set; see “*bx\_ctgentype*” on page 217.

**See also** “*BestXCTGenDefaultSet*” on page 85  
     “*BestXCTGenGet*” on page 85  
     “*BestXExerciserProg*” on page 61

# Completer-Initiator Programming Functions

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

Function	Action
<i>"BestXCIGenDefaultSet" on page 92</i>	Sets all generic completer-initiator properties to default values.
<i>"BestXCIGenSet" on page 93</i>	Sets generic completer-initiator properties.
<i>"BestXCIGenGet" on page 92</i>	Gets generic completer-initiator properties.
<i>"BestXCIBehMemInit" on page 90</i>	Initializes the completer-initiator behavior memory.
<i>"BestXCIBehDefaultSet" on page 88</i>	Sets all completer-initiator behavior properties to default values.
<i>"BestXCIBehSet" on page 91</i>	Sets a completer-initiator behavior property on one line of the behavior memory.
<i>"BestXCIBehGet" on page 89</i>	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 213.

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 230.

**Input Parameters** **handle** Session identification.

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

**See also** “*BestXCIBehSet*” on page 91  
“*BestXCIBehGet*” on page 89  
“*BestXExerciserProg*” on page 61



## BestXCIBehGet

**Call**    `bx_errtype BestXCIBehGet (`  
               `bx_handletype    handle,`  
               `bx_int32            offset,`  
               `bx_cibehdtype    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 230.

**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\_cibehdtype*” on page 213.

**Output Parameters**    **val**    Value of the completer-initiator behavior property; see “*bx\_cibehdtype*” on page 213.

**See also**    “*BestXCIBehDefaultSet*” on page 88  
               “*BestXCIBehSet*” on page 91

## 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\_cibehdtype*” on page 213.

**CLI Equivalent** `BestXCIBehMemInit`

**CLI Abbreviation** `cibehmeminit`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXRIBehMemInit*” on page 67  
“*BestXRTBehMemInit*” on page 76  
“*BestXCTBehMemInit*” on page 83

## 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 230.

**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 213.

**val** Value of the completer-initiator behavior property to be set; see “*bx\_cibehtype*” on page 213.

**See also** “*BestXCIBehDefaultSet*” on page 88  
     “*BestXCIBehGet*” on page 89  
     “*BestXExerciserProg*” on page 61

## 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 215.

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 230.

**Input Parameters** **handle** Session identification.

**See also** “BestXCIGenSet” on page 93  
 “BestXCIGenGet” on page 92  
 “BestXExerciserProg” on page 61

## 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 230.

**Input Parameters** **handle** Session identification.

**prop** Property to be obtained; see “*bx\_cigentype*” on page 215.

**Output Parameters** **val** Property value; see “*bx\_cigentype*” on page 215.

**See also** “BestXCIGenDefaultSet” on page 92  
 “BestXCIGenSet” on page 93

## 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 230.

**Input Parameters**    **handle**    Session identification.

**prop**    Property to be set; see “*bx\_cigentype*” on page 215.

**val**    Value to which the property is set; see “*bx\_cigentype*” on page 215.

**See also**    “*BestXCIGenDefaultSet*” on page 92  
                  “*BestXCIGenGet*” on page 92  
                  “*BestXExerciserProg*” on page 61

# Configuration Space Programming Functions

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

Function	Action
<i>"BestXConfRegSet" on page 97</i>	Sets a configuration space register.
<i>"BestXConfRegGet" on page 94</i>	Gets a configuration space register.
<i>"BestXConfRegMaskSet" on page 96</i>	Sets a configuration mask register.
<i>"BestXConfRegMaskGet" on page 95</i>	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 230*.

**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 97*

## 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 230.

**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 96

## 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 230.

**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 95  
 “*BestXExerciserProg*” on page 61



## 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 210.

### 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 230.

**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 94  
 “*BestXExerciserProg*” on page 61

# Decoder Programming Functions

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

Function	Action
<i>"BestXTDecoderDefaultSet" on page 100</i>	Sets all target decoder properties to default values.
<i>"BestXTDecoderAllSet" on page 99</i>	Sets all target decoder properties.
<i>"BestXTDecoderSet" on page 102</i>	Sets a target decoder property.
<i>"BestXTDecoderGet" on page 101</i>	Gets a target decoder property.
<i>"BestXCTSplitCondDefaultSet" on page 103</i>	Sets all split decoder properties to default values.
<i>"BestXCTSplitCondSet" on page 105</i>	Sets a split decoder property.
<i>"BestXCTSplitCondGet" on page 104</i>	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 230.

**Input Parameters** **handle** Session identification.

**dec** Target decoder to be set. Valid properties are bar0 ... bar5. See “*bx\_dectype*” on page 221.

**location** Location for the requested address range to be set. See “*bx\_decproptype*” on page 220.

**size** Decoder size to be set. See “*bx\_decproptype*” on page 220.

**prefetch** Defines if the memory is prefetchable. See “*bx\_decproptype*” on page 220.

**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 220.

**resbase** Internal resource base address to be set. See “*bx\_decproptype*” on page 220.

**ressize** Size of the internal resource to be set. See “*bx\_decproptype*” on page 220.

**compare** Defines whether the data compare is switched on for this decoder. See “*bx\_decproptype*” on page 220.

**See also** “*BestXTDecoderDefaultSet*” on page 100  
 “*BestXTDecoderGet*” on page 101  
 “*BestXTDecoderSet*” on page 102  
 “*BestXExerciserProg*” on page 61

## 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 $2^{20} \dots 2^{16}$	
BAR 4 and 5	- I/O decoder with size 64 K - resource base $2^{20} \dots 2^{16}$	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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXTDecoderGet*” on page 101  
 “*BestXTDecoderSet*” on page 102  
 “*BestXExerciserProg*” on page 61  
 “*BestXTDecoderAllSet*” on page 99

## 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 230.

**Input Parameters**    **handle**    Session identification.

**dec**    Target decoder to be selected; see “*bx\_dectype*” on page 221.

**prop**    Target decoder property; see “*bx\_decproptype*” on page 220.

**Output Parameters**    **val**    Property value; see “*bx\_decproptype*” on page 220.

**See also**    “*BestXTDecoderDefaultSet*” on page 100  
               “*BestXTDecoderSet*” on page 102  
               “*BestXTDecoderAllSet*” on page 99

## 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 230.

**Input Parameters**   **handle**    Session identification.

**dec**        Target decoder to be selected; see “*bx\_dectype*” on page 221.

**prop**      Target decoder property; see “*bx\_decproptype*” on page 220.

**val**        Property value to be set; see “*bx\_decproptype*” on page 220.

**See also**        “*BestXTDecoderDefaultSet*” on page 100

                  “*BestXTDecoderGet*” on page 101

                  “*BestXExerciserProg*” on page 61

                  “*BestXTDecoderAllSet*” on page 99

## 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\_ctsplitttype*” on page 218.

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 230.

**Input Parameters**    **handle**    Session identification.

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

**See also**    “*BestXCTSplitCondGet*” on page 104

“*BestXCTSplitCondSet*” on page 105

“*BestXExerciserProg*” on page 61

## BestXCTSplitCondGet

**Call**    `bx_errtype BestXCTSplitCondGet(  
                   bx_handletype    handle,  
                   bx_int32        dec,  
                   bx_ctsplitttype 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**    `ctsplitsplitcondget dec=<dec> prop=<prop>`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**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\_ctsplitttype*” on page 218.

**Output Parameters**    **val**    Property value; see “*bx\_ctsplitttype*” on page 218.

**See also**    “*BestXCTSplitCondDefaultSet*” on page 103  
               “*BestXCTSplitCondSet*” on page 105



## 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 230.

**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 218.

**val** Property value to be set; see “*bx\_ctsplittype*” on page 218.

**See also** “*BestXCTSplitCondDefaultSet*” on page 103  
 “*BestXCTSplitCondGet*” on page 104  
 “*BestXExerciserProg*” on page 61

# Expansion ROM Programming Functions

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

Function	Action
<i>"BestXExpRomWrite" on page 107</i>	Writes a byte to the expansion ROM.
<i>"BestXExpRomRead" on page 106</i>	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 230*.

**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  $\leq 65536$ .

**Output Parameters** **data** Return value.

**Example** `BestXExpRomRead offset=0 numbytes=8`

**See also** *"BestXExpRomWrite" on page 107*

## 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 230.

**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 106

# Data Memory Functions

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

Function	Action
<i>"BestXDataMemInit" on page 108</i>	Initializes the data memory of the testcard by filling it with zeros.
<i>"BestXDataMemWrite" on page 110</i>	Writes to the data memory of the testcard.
<i>"BestXDataMemRead" on page 109</i>	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 230*.

**Input Parameters** **handle** Session identification.

**See also** *"BestXDataMemRead" on page 109*  
*"BestXDataMemWrite" on page 110*

## 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 (8000h). 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 230.

**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 108  
 “*BestXDataMemWrite*” on page 110

## 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 230.

**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 108  
 “*BestXDataMemRead*” on page 109

# Host Access Functions

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

Function	Action
<i>"BestXHostPCIRegWrite" on page 113</i>	Writes a register of a PCI-X device.
<i>"BestXHostPCIRegRead" on page 112</i>	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** `hostpciiregread addrspace=<addrspace> addr=<addr> size=<size>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**space** Type of address space for the register access; see “*bx\_addrspacetype*” on page 209.

**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 252.

**Output Parameters** **val** Value obtained from the register.

**See also** “*BestXHostPCIRegWrite*” on page 113



## 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 230.

**Input Parameters** **handle** Session identification.

**space** Type of address space for the register access; see “*bx\_addrspacetype*” on page 209.

**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 252.

**See also** “*BestXHostPCIRegRead*” on page 112  
 “*BestXExerciserProg*” on page 61



# Analyzer Functions

The PCI-X analyzer functions are divided as follows:

Analyzer Functions	Action
Pattern Term Function	Sets pattern terms that can be used as input for the trigger sequencer and the performance counters.
Trigger Sequencer Programming Functions	Set up the memory-based state machine.
Trace Memory Functions	Read out, display or postprocess the complete content of the trace memory.
Performance Sequencer Programming Functions	Program the performance counters for selected performance measurements.

## Pattern Term Function

The following function is used for the pattern terms:

Function	Action
<i>"BestXPattProg" on page 116</i>	Defines one of the pattern recognizers.

## BestXPattProg

**Call** `bx_errtype BestXPattProg(  
           bx_handletype  handle,  
           bx_patttype    prop,  
           bx_charptrtype pattern );`

**Description** Sets a pattern into one of the pattern recognizers, which can be used as input for the trigger sequencer and for the performance counters.

Patterns can be divided into:

- bus pattern
- observer pattern
- error pattern

Each pattern is built by using specific signals that must be logically combined.

**CLI Equivalent** `BestXPattProg prop=<prop> pattern=<pattern>`

**CLI Abbreviation** `pattprog prop=<prop> pattern=<pattern>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**prop** Pattern recognizer that is used with the particular pattern; see “*bx\_patttype*” on page 235.

**pattern** Logical expression for the condition. The string must be set in quotation marks. It is built by using specific signals (“*bx\_signaltype*” on page 247) and combining them logically. To build pattern terms see:

- “*Pattern Term Operators*” on page 117
- “*Syntax Diagrams for Numbers and 10XNumbers*” on page 117
- “*Bus Pattern Term Syntax Diagram*” on page 118
- “*Error Pattern Term Syntax Diagram*” on page 120

**See also** “*BestXTrigCondSet*” on page 122

## Pattern Term Operators

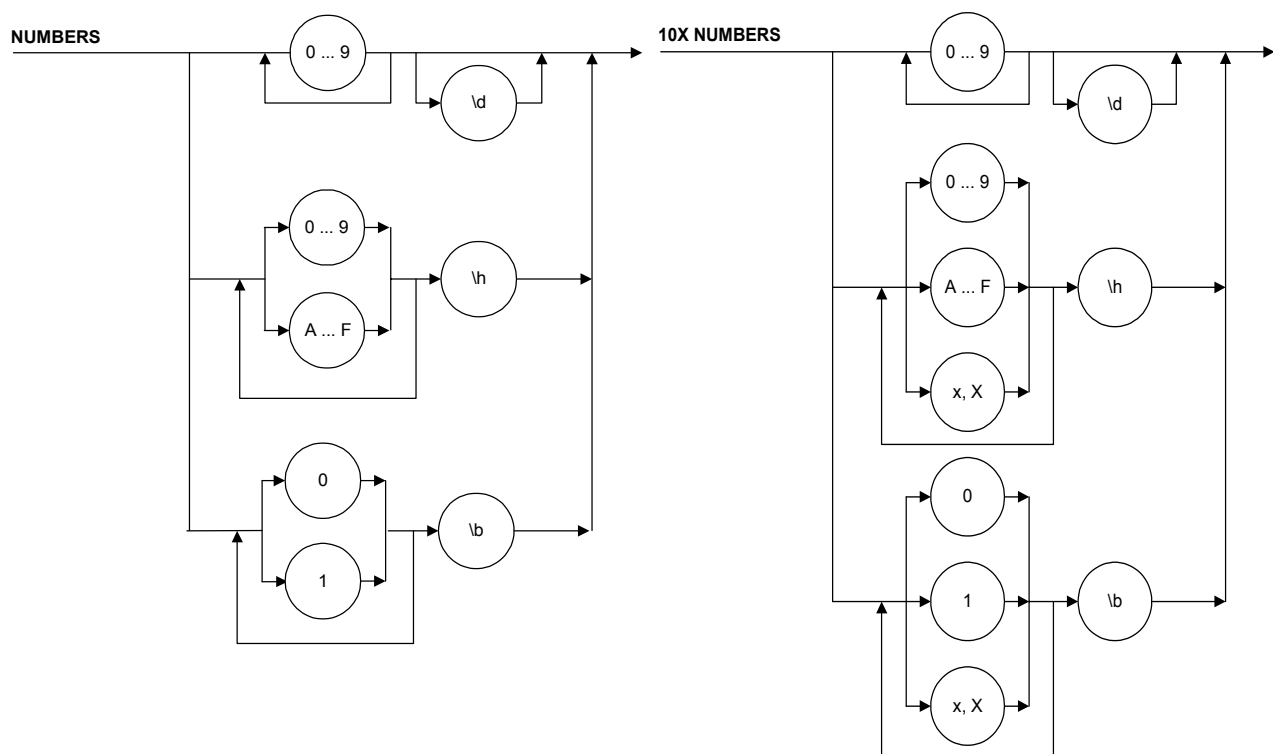
The following table shows the operators that can be used in pattern terms to combine the signal labels:

Operator	Operation	Pattern Term		
		Bus	Observer	Error
!	negation	X	X	X
==	compare for equality	X	–	–
&&	logical AND	X	X	–
	logical OR	–	–	X

Signals of “10X vector type” can be queried bitwise; see the examples in “*Bus Pattern Term Syntax Diagram*” on page 118.

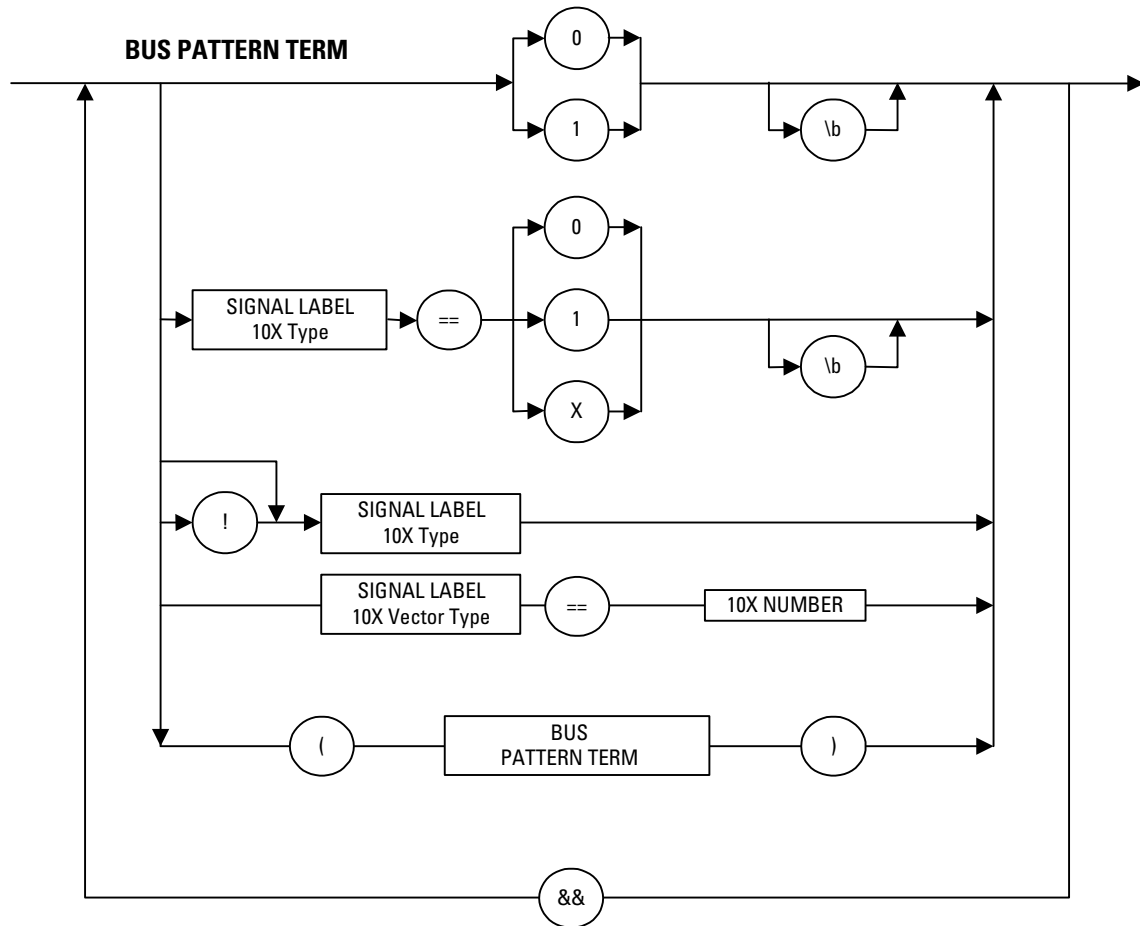
## Syntax Diagrams for Numbers and 10XNumbers

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



These numbers can be used when building logical expressions to program all pattern terms.

To see which signals can be used for the bus pattern, refer to *“bx\_signaltype” on page 247*.

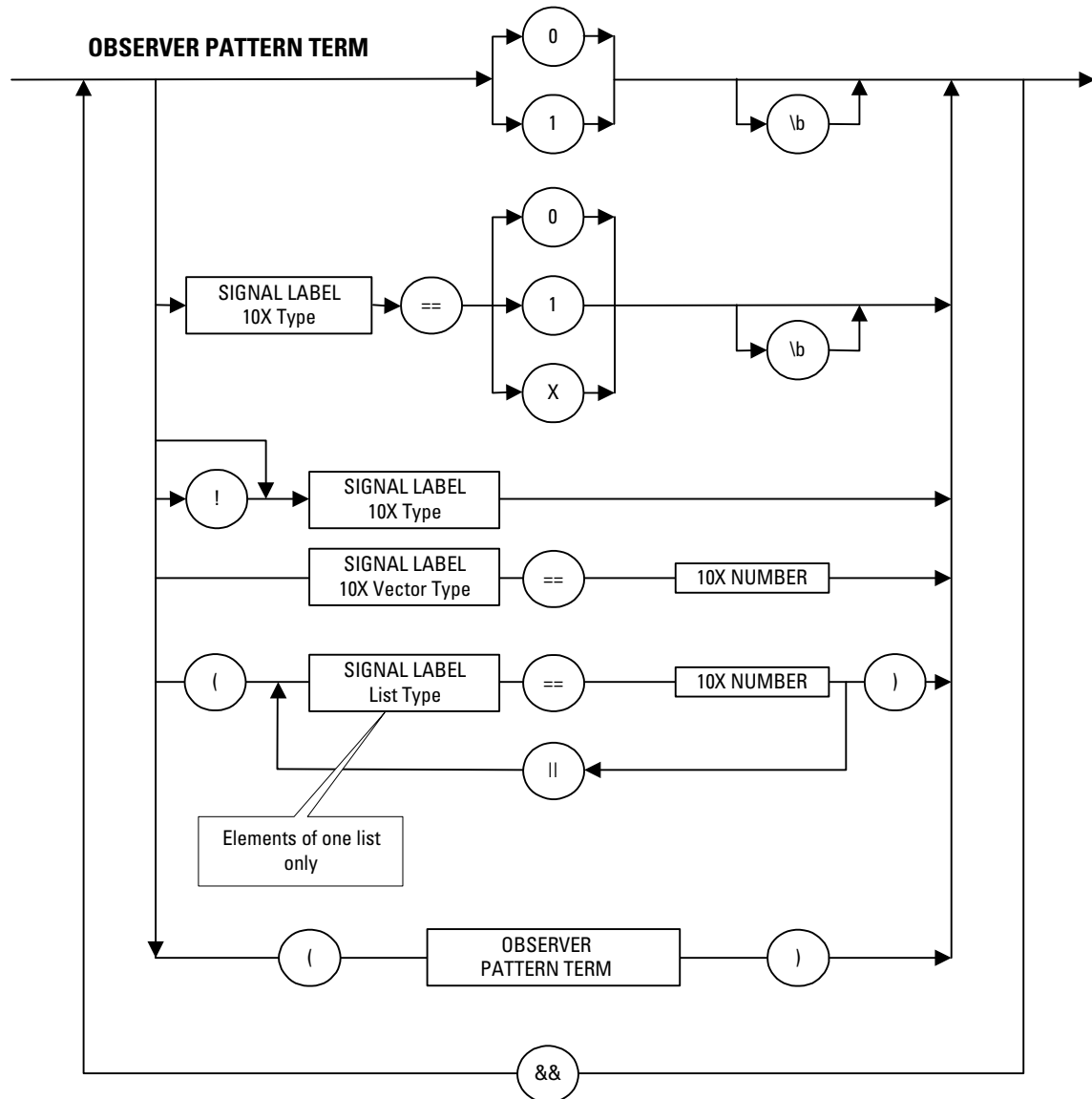


- " !IRDY && !TRDY"

- `"!IRDY && !TRDY"`  
IRDY low and TRDY low.
- `"addr_phase==1 && AD32==b8xxx\h"`  
Address phase that addresses the range b8000 ... b8fff.

## Observer Pattern Term Syntax Diagram

To see which signals can be used for the bus pattern and the observer pattern, refer to “*bx\_signaltype*” on page 247.



### Examples:

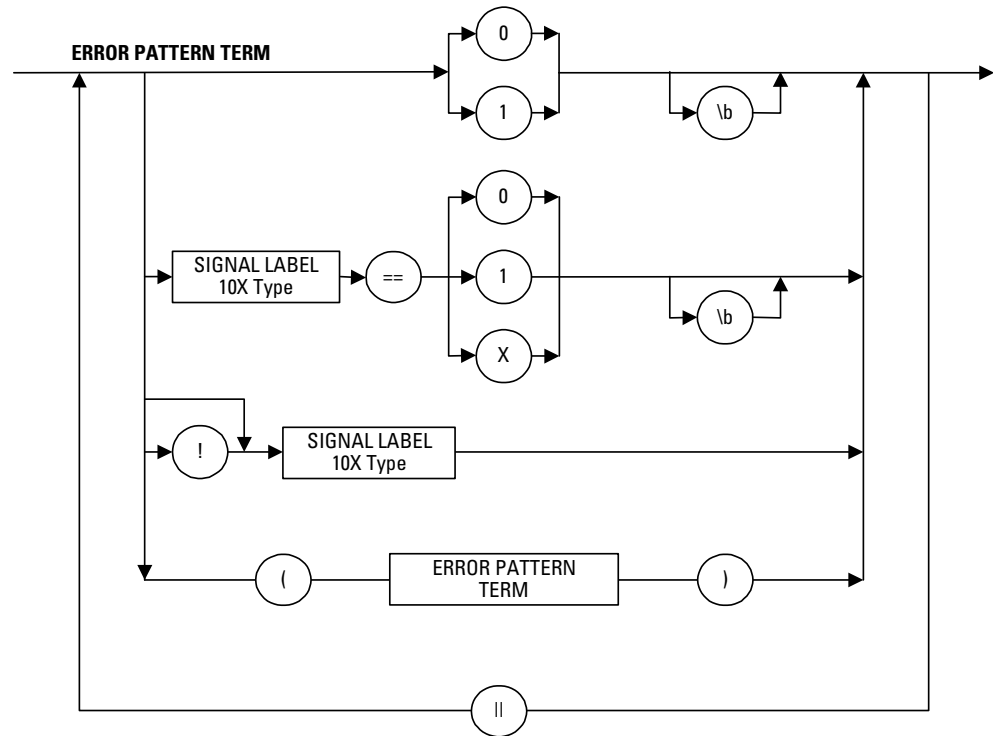
- "(xact\_cmd==6 || xact\_cmd==7) && ri\_act==1"

The requester-initiator executes a Memory Read DWord or a Memory Write command.

- "xact\_attr\_ad==xx0008xx\h && bstate==8"

The initiator (identification: bus 0, device 1, function 0) transfers data.

To see which signals can be used for the error pattern, refer to *"bx\_signaltype"* on page 247.



```
"proterr || dcomperr"
```

Detects whether a data compare or a protocol error occurred.



# Trigger Sequencer Programming Functions

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

Function	Action
<i>"BestXTrigGenDefaultSet" on page 125</i>	Sets the preload values of the feedback counters to the default value.
<i>"BestXTrigGenGet" on page 126</i>	Gets the preload value of a feedback counter.
<i>"BestXTrigGenSet" on page 127</i>	Sets the preload value of a feedback counter.
<i>"BestXTrigDefaultSet" on page 125</i>	Sets the entire trigger sequencer to a single-state machine with harmless behavior.
<i>"BestXTrigTranCondDefaultSet" on page 129</i>	Sets all properties of a line to defaults.
<i>"BestXTrigTranSet" on page 130</i>	Sets a numeric transition property ("state" or "next state").
<i>"BestXTrigCondSet" on page 122</i>	Sets a condition for the trigger sequencer.
<i>"BestXTrigProg" on page 128</i>	Writes transition and transition conditions to the trigger sequencer memory.

## BestXTrigCondSet

**Call** `bx_errtype BestXTrigCondSet (`  
     `bx_handletype      handle,`  
     `bx_int32            offset,`  
     `bx_trigcondtype    prop,`  
     `bx_charptrtype     cond );`

**Description** Sets a condition for the selected line in the trigger sequencer description table.

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

To write the condition to the testcard, use the `BestXTrigProg` call.

**CLI Equivalent** `BestXTrigCondSet offset=<offset> prop=<prop> cond=<cond>`

**CLI Abbreviation** `trigcondset offset=<offset> prop=<prop> cond=<cond>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**offset** Number (0 ... 127).

**prop** Condition property to be set; see “*bx\_trigcondtype*” on page 259.

**cond** Condition string. The string must be set in quotation marks. See “*Conditions Reference*” on page 123.

**See also** “*BestXTrigGenDefaultSet*” on page 125

“*BestXTrigTranSet*” on page 130

“*BestXTrigProg*” on page 128

## Conditions Reference

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

These condition strings can consist of:

- All pattern terms; see *“bx\_patttype” on page 235*.  
You can define the pattern terms with *“BestXPattProg” on page 116*.
- The terminal count of the sequencer feedback counters.
- Logical operators (logical AND, logical OR, ...).
- True or false settings (“1” or “0”).

### Example:

Valid condition string to program the trigger sequencer:

```
"( !BX_PATT_BUS0 || BX_PATT_BUS1 || BX_PATT_OBS0 )"
```

### Identifiers for Sequencers

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

Identifier	Description
bus0 bus1	BX_PATT_BUS0, BX_PATT_BUS1
obs0 obs1 obs2 obs3	BX_PATT_OBS0, BX_PATT_OBS1, BX_PATT_OBS2, BX_PATT_OBS3
err0	BX_PATT_ERR0
cond0 cond1	BX_PATT_CONDO, BX_PATT_COND1
tc_fba	Terminal count of the trigger sequencer and performance sequencer feedback counters.
tc_fbb	Terminal count of the trigger sequencer feedback counter.

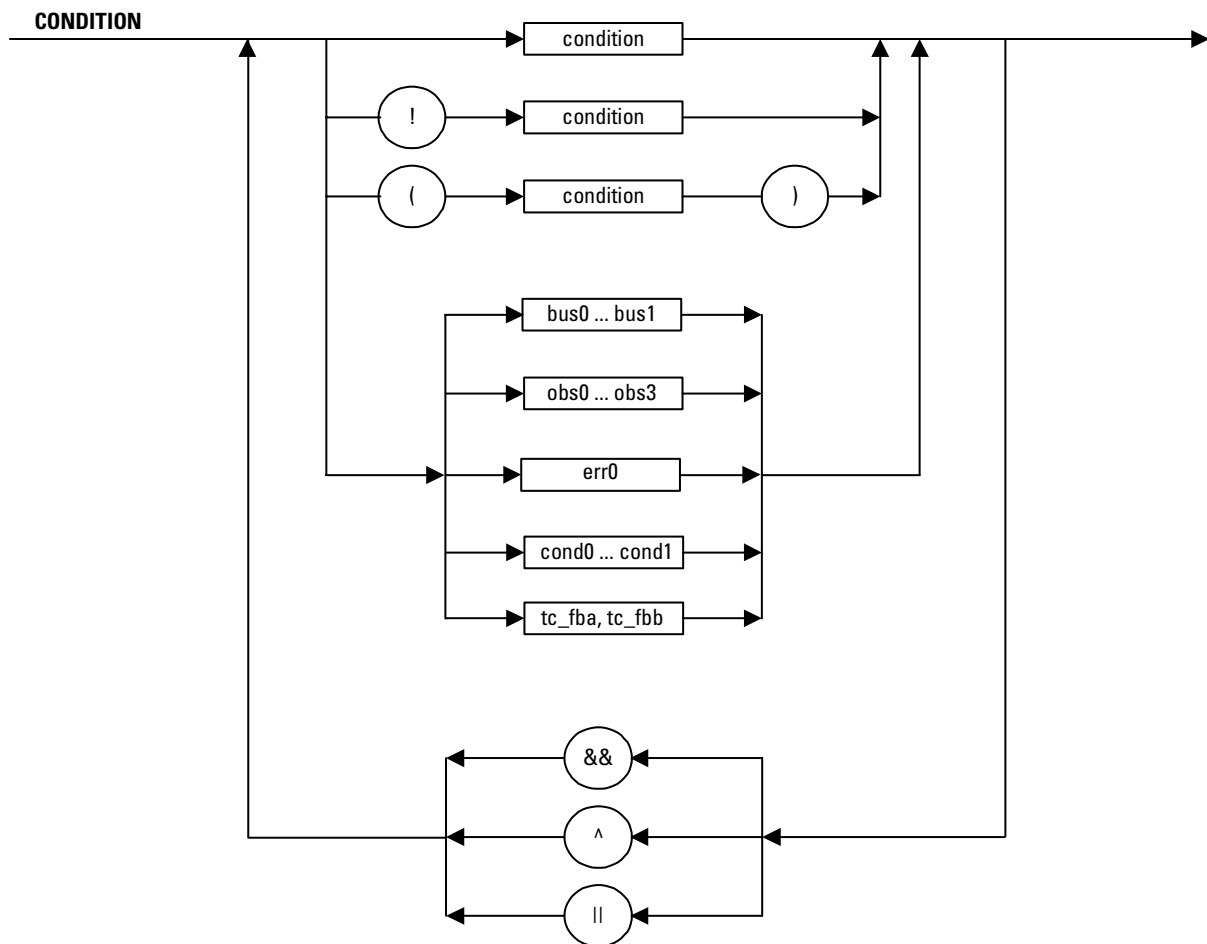
**Logical Operators** The following table shows the logical operators in the order of their priority:

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

“(“ and “)” can be used to override priorities.

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

**NOTE** The terminal count `tc_fbb` is only available for the trigger sequencer.



## BestXTrigDefaultSet

**Call** `bx_errtype BestXTrigDefaultSet( bx_handletype handle );`

**Description** Initializes the entire trace memory trigger sequencer to a single-state machine with harmless behavior. All previously entered lines are cleared.

**CLI Equivalent** `BestXTrigDefaultSet`

**CLI Abbreviation** `trigdefset`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXTrigGenDefaultSet*” on page 125

## BestXTrigGenDefaultSet

**Call** `bx_errtype BestXTrigSeqGenDefaultSet( bx_handletype handle );`

**Description** Sets all generic trigger sequencer properties to their default values. The properties determine the preload values of the feedback counters A and B for the trace memory trigger sequencer. For the default values, see “*bx\_triggentype*” on page 259.

**CLI Equivalent** `BestXTrigGenDefaultSet`

**CLI Abbreviation** `triggendefset`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXTrigGenSet*” on page 127

“*BestXTrigGenGet*” on page 126

## BestXTrigGenGet

**Call**    `bx_errtype BestXTrigGenGet (`  
                  `bx_handletype    handle,`  
                  `bx_triggentype   prop,`  
                  `bx_int32           *val );`

**Description**   Gets the value of a generic trigger sequencer property. The properties determine the preload values of the feedback counters A and B for the trace memory trigger sequencer.

**CLI Equivalent**   `BestXTrigGenGet prop=<prop>`

**CLI Abbreviation**   `triggenget prop=<prop>`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**   **handle**    Session identification.

**prop**    Property to be obtained; see “*bx\_triggentype*” on page 259.

**Output Parameters**   **val**    Value of the property; see “*bx\_triggentype*” on page 259.

**See also**    “*BestXTrigGenDefaultSet*” on page 125  
                  “*BestXTrigGenSet*” on page 127

## BestXTrigGenSet

**Call** `bx_errtype BestXTrigGenSet (`  
     `bx_handletype     handle,`  
     `bx_triggentype    prop,`  
     `bx_int32           val );`

**Description** Sets the value of the generic trigger sequencer property. Generic properties determine the preload values of the feedback counters A and B for the trace memory trigger sequencer.

**CLI Equivalent** `BestXTrigGenSet prop=<prop> val=<val>`

**CLI Abbreviation** `triggerset prop=<prop> val=<val>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**prop** Property to be set; see “*bx\_triggentype*” on page 259.

**val** Value to which the property is set; see “*bx\_triggentype*” on page 259.

**See also** “*BestXTrigGenDefaultSet*” on page 125  
           “*BestXTrigGenGet*” on page 126

## BestXTrigProg

**Call** `bx_errtype BestXTrigProg( bx_handletype handle );`

**Description** Writes the transition and transition condition properties that were previously set in the trigger sequencer description table to the trigger sequencer memory.

This function also checks for syntax errors and logical consistency within the programmed state machine. If any are found, the function returns an error.

**CLI Equivalent** `BestXTrigProg`

**CLI Abbreviation** `trigprog`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXTrigCondSet*” on page 122  
“*BestXTrigTranSet*” on page 130



## BestXTrigTranCondDefaultSet

**Call** `bx_errtype BestXTrigTranCondDefaultSet (`  
    `bx_handletype handle,`  
    `bx_int32 offset );`

**Description** Sets all properties for a specific offset in the trigger sequencer description table to default values.

For a description of properties and default values, refer to “*bx\_trigtrantype*” on page 260 and “*bx\_trigcondtype*” on page 259.

To write these settings to the testcard, use the BestXTrigProg call.

**CLI Equivalent** `BestXTrigTranCondDefaultSet offset=<offset>`

**CLI Abbreviation** `trigtranconddefset offset=<offset>`

**Return Value** Error code; see “*bx\_errtype*” on page 230

**Input Parameters** **handle** Session identification.

**offset** Number (0 ... 127).

**See also** “*BestXTrigTranSet*” on page 130  
“*BestXTrigCondSet*” on page 122  
“*BestXTrigProg*” on page 128

## BestXTrigTranSet

**Call** `bx_errtype BestXTrigTranSet (`  
     `bx_handletype handle,`  
     `bx_int32 offset,`  
     `bx_trigtrantype prop,`  
     `bx_int32 val );`

**Description** Sets a numeric transition property (“state” or “next state”) for the trigger sequencer. To write the property to the testcard, use the `BestXTrigProg` call.

**CLI Equivalent** `BestXTrigTranSet offset=<offset> prop=<prop> val=<val>`

**CLI Abbreviation** `trigtranaset offset=<offset> prop=<prop> val=<val>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**offset** Number (0 ... 127).

**prop** Property to be set; see “*bx\_trigtrantype*” on page 260.

**val** Value to which the property is to be set; see “*bx\_trigtrantype*” on page 260.

**See also** “*BestXTrigGenDefaultSet*” on page 125  
           “*BestXTrigCondSet*” on page 122  
           “*BestXTrigProg*” on page 128

# Trace Memory Functions

The following functions are used for the trace memory:

Function	Action
<i>"BestXTraceDefaultWrite" on page 135</i>	Sets all trace memory properties to default values.
<i>"BestXTraceWrite" on page 139</i>	Writes a property for the trace memory.
<i>"BestXTraceRead" on page 137</i>	Gets a property for the trace memory.
<i>"BestXTraceRun" on page 137</i>	Runs the trigger and storage sequencer.
<i>"BestXTraceStop" on page 138</i>	Stops the trigger and storage sequencer.
<i>"BestXTraceDataRead" on page 134</i>	Loads trace memory lines from the testcard.
<i>"BestXTraceBitPosGet" on page 132</i>	Returns the position and length of a signal in a trace memory line.
<i>"BestXTraceBytePerLineGet" on page 133</i>	Returns the number of bytes of a trace memory line.
<i>"BestXPattProg" on page 116</i>	Defines a pattern recognizer.
<i>"BestXTraceDump" on page 136</i>	Writes the complete trace memory content to a file.

## BestXTraceBitPosGet

**Call** `bx_errtype BestXTraceBitPosGet(  
     bx_handletype handle,  
     bx_signaltype signal,  
     bx_int32 *pos,  
     bx_int32 *length );`

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

**CLI Equivalent** `BestXTraceBitPosGet signal=<signal>`

**CLI Abbreviation** `tracebitposget signal=<signal>`

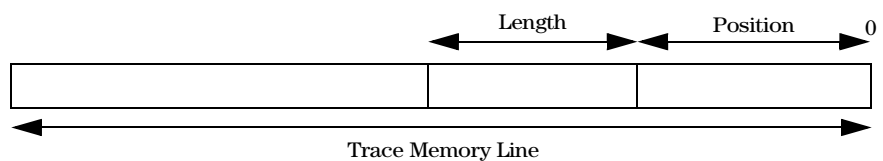
**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**signal** Signal for which the position and length are to be determined; see “*bx\_signaltype*” on page 247.

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

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



**See also** “*BestXTraceBytePerLineGet*” on page 133

## BestXTraceBytePerLineGet

**Call**    `bx_errtype BestXTraceBytePerLineGet (`  
          `bx_handletype    handle,`  
          `bx_int32            *bytes_per_line );`

**Description**    Returns the number of bytes in a trace memory line.

**CLI Equivalent**    `BestXTraceBytePerLineGet`

**CLI Abbreviation**    `tracebyteperlineget`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**    **handle**    Session identification.

**Output Parameters**    **bytes\_per\_line**    Number of bytes per line of trace data.

**See also**    “*BestXTraceDataRead*” on page 134

## BestXTraceDataRead

**Call** `bx_errtype BestXTraceDataGet (`  
     `bx_handletype handle,`  
     `bx_int32 offset,`  
     `bx_int32 num,`  
     `bx_int32 *data );`

**Description** Reads trace memory lines from the testcard. Before you read the data, you must allocate a data array that is large enough to hold the data. The required size of the data array can be calculated in the following way:

- Size of the array in bytes =  
     number of lines (num) × number of bytes per line;  
     (the number of bytes per line can be requested with  
     “*BestXTraceBytePerLineGet*” on page 133)
- Size of the array in dwords =  
     size of the array in bytes / 4

**CLI Equivalent** `BestXTraceDataRead offset=<offset> num=<num>`

**CLI Abbreviation** `tracedataread offset=<offset> num=<num>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**offset** First line to be read from trace memory.

**num** Number of lines to be read from trace memory.

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

**See also** “*BestXTraceBytePerLineGet*” on page 133

## BestXTraceDefaultWrite

**Call** `bx_errtype BestXTraceDefaultWrite( bx_handletype handle );`

**Description** Writes the trigger counter preload value for the trace memory to the default value.

For the default value, see “*bx\_tracetype*” on page 258.

**CLI Equivalent** `BestXTraceDefaultWrite`

**CLI Abbreviation** `tracedefwrite`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXTraceWrite*” on page 139  
“*BestXTraceRead*” on page 137

## BestXTraceDump

**Call**    `bx_errtype BestXTraceDump(  
          bx_handletype    handle,  
          bx_charptrtype   file );`

**Description**    Writes the whole content of the trace memory to a file.

**NOTE**    It may take a considerable amount of time to store the data to a file, especially if the connection is slow.

**CLI Equivalent**    `BestXTraceDump file=<file>`

**CLI Abbreviation**    `tracedump file=<file>`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**    **handle**    Session identification.

**file**    Name and complete path of the file.

**See also**    “*BestXTraceDefaultWrite*” on page 135  
              “*BestXTraceRead*” on page 137



## BestXTraceRead

**Call** `bx_errtype BestXTraceRead(  
           bx_handletype  handle,  
           bx_tracetype  prop,  
           bx_int32      *val );`

**Description** Reads the trigger counter preload value for the trace memory.

**CLI Equivalent** `BestXTraceRead prop=<prop>`

**CLI Abbreviation** `traceread prop=<prop>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**prop** Property to be read; see “*bx\_tracetype*” on page 258.

**Output Parameters** **val** Value of the property to be read; see “*bx\_tracetype*” on page 258.

**See also** “*BestXTraceRead*” on page 137  
 “*BestXTraceDefaultWrite*” on page 135

## BestXTraceRun

**Call** `bx_errtype BestXTraceRun( bx_handletype handle );`

**Description** Runs the trigger and storage sequencer. Data is acquired according to the trace memory trigger sequencer. The trace memory of the piggyback board will be filled or data will be transferred through the Logic Analyzer connection.

**CLI Equivalent** `BestXTraceRun`

**CLI Abbreviation** `tracerun`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXTraceStop*” on page 138

## BestXTraceStop

**Call** `bx_errtype BestXTraceStop( bx_handletype handle );`

**Description** Stops the trigger and storage sequencer. The trace memory stops acquiring data. If a Logic Analyzer is connected, any data transfer to it stops.

**CLI Equivalent** `BestXTraceStop`

**CLI Abbreviation** `tracestop`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** –

## BestXTraceWrite

**Call**    `bx_errtype BestXTraceWrite(  
          bx_handletype  handle,  
          bx_tracetype  prop,  
          bx_int32       val );`

**Description**    Writes the trigger counter preload value for the trace memory.

**CLI Equivalent**    `BestXTraceWrite prop=<prop> val=<val>`

**CLI Abbreviation**    `tracewrite prop=<prop> val=<val>`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**    **handle**    Session identification.

**prop**    Property to be set; see “*bx\_tracetype*” on page 258.

**val**    Value to which the property is set; see “*bx\_tracetype*” on page 258.

**See also**    “*BestXTraceDefaultWrite*” on page 135  
              “*BestXTraceRead*” on page 137

# Performance Sequencer Programming Functions

**Setup and Programming** The functions of this category are used to set up and define the performance measurements and the corresponding sequencers:

Function	Action
<i>"BestXPerfGenDefaultSet" on page 144</i>	Sets the generic performance properties to default values.
<i>"BestXPerfGenSet" on page 146</i>	Sets the value of a generic performance measurement property.
<i>"BestXPerfGenGet" on page 145</i>	Gets the value of a generic performance measurement property.
<i>"BestXPerfDefaultSet" on page 143</i>	Initializes the entire trace memory trigger sequencer to a single-state machine.
<i>"BestXPerfTranCondDefaultSet" on page 148</i>	Sets all properties of a line to default values.
<i>"BestXPerfTranSet" on page 149</i>	Sets a numeric transition property ("state" or "next state").
<i>"BestXPerfCondSet" on page 141</i>	Sets a transition condition for the trigger sequencer of a performance measurement.
<i>"BestXPerfProg" on page 147</i>	Writes the transition and transition conditions of a performance measurement to the performance sequencer memory.

**Running and Evaluating** The following functions are used to run and evaluate the measurement:

Function	Action
<i>"BestXPerfRun" on page 148</i>	Starts all counters.
<i>"BestXPerfStop" on page 150</i>	Stops all counters.
<i>"BestXPerfUpdate" on page 150</i>	Captures the values of all performance counters.
<i>"BestXPerfCtrRead" on page 142</i>	Reads counter values.

## BestXPerfCondSet

**Call** `bx_errtype BestXPerfCondSet (`  
     `bx_handletype handle,`  
     `bx_int32 meas,`  
     `bx_int32 offset,`  
     `bx_perfcondtype prop,`  
     `bx_charptrtype condition );`

**Description** Sets a condition in the sequencer description table for a specific measurement and a specific line.

The conditions are:

- transition condition
- conditions to increment nominator or denominator counter
- conditions to decrement or preload the feedback counter

To write the condition to the testcard, use the BestXPerfProg call.

**CLI Equivalent** `BestXPerfCondSet meas=<meas> offset=<offset> prop=<prop>`  
`condition=<condition>`

**CLI Abbreviation** `perfcondset meas=<meas> offset=<offset> prop=<prop>`  
`condition=<condition>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**meas** Value to identify the measurement; see “*Measurement Identifier*” on page 142.

**offset** Number (0 ... 127).

**prop** Property to be set; see “*bx\_perfcondtype*” on page 235.

**condition** Condition string to which the property is set. The string must be set in quotation marks. See “*Conditions Reference*” on page 123.

**See also** “*BestXPerfTranCondDefaultSet*” on page 148  
 “*BestXPerfTranSet*” on page 149  
 “*BestXPerfProg*” on page 147

## BestXPerfCtrRead

**Call** `bx_errtype BestXPerfCtrRead(  
     bx_handletype handle,  
     bx_int32 meas,  
     bx_int32 counter,  
     bx_int32 *val );`

**Description** Reads the counter values of a performance measurement. Because all counters consist of 64 bits, you need two accesses to read the lower and the upper 32 bits of each counter.

**NOTE** The counter values must have previously been updated with “*BestXPerfUpdate*” on page 150.

**CLI Equivalent** `BestXPerfCtrRead meas=<meas> counter=<counter>`

**CLI Abbreviation** `perfctrread meas=<meas> counter=<counter>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**meas** Value to identify the measurement; see “*Measurement Identifier*” on page 142.

**counter** Value to identify the counter; see “*Counter Identifier*” on page 143.

**Output Parameters** **val** Value of the queried counter.

**See also** –

## Measurement Identifier

Measurement Identifier	Description
BX_PERFMEAS_0, BX_PERFMEAS_1, BX_PERFMEAS_2, BX_PERFMEAS_3	Selects one of the performance measurements.  For E2929A testcards, only the measurements 0 and 1 are available.

## Counter Identifier

Counter Identifier	Description
BX_PERFCTR_A BX_PERFCTR_A_HIGH	Nominator counter value (lower and upper 32 bits).
BX_PERFCTR_B BX_PERFCTR_B_HIGH	Denominator counter value (lower and upper 32 bits).
BX_PERFCTR_REFERENCE BX_PERFCTR_REFERENCE_HIGH	PCI-X clock reference counter value (lower and upper 32 bits). This counter is always present.

## BestXPerfDefaultSet

**Call** `bx_errtype BestXPerfDefaultSet(  
    bx_handletype handle,  
    bx_int32 meas );`

**Description** Initializes the entire trace memory trigger sequencer to a single-state machine with harmless behavior. All previously entered lines are cleared. For a description of properties and default values, see “*bx\_perfrantype*” on page 236 and “*bx\_perfcondtype*” on page 235.

**CLI Equivalent** `BestXPerfDefaultSet meas=<meas>`

**CLI Abbreviation** `perfdefset meas=<meas>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**meas** Value to identify the measurement; see “*Measurement Identifier*” on page 142.

**See also** “*BestXPerfTranCondDefaultSet*” on page 148

## BestXPerfGenDefaultSet

**Call** `bx_errtype BestXPerfGenDefaultSet (`  
           `bx_handletype handle,`  
           `bx_int32 meas );`

**Description** Sets the generic performance properties of a performance measurement to default values. These properties are used to:

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

For generic properties and default values, see “*bx\_perfgentype*” on page 236.

**CLI Equivalent** `BestXPerfGenDefaultSet meas=<meas>`

**CLI Abbreviation** `perfgendefset meas=<meas>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**meas** Value to identify the measurement; see “*Measurement Identifier*” on page 142.

**See also** “*BestXPerfGenGet*” on page 145  
 “*BestXPerfGenSet*” on page 146



## BestXPerfGenGet

**Call**    `bx_errtype BestXPerfGenGet (`  
              `bx_handletype    handle,`  
              `bx_int32            meas,`  
              `bx_perfgentype    prop,`  
              `bx_int32            *val );`

**Description**    Gets the value of a generic performance property of a performance measurement. These properties are used to:

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

**CLI Equivalent**    `BestXPerfGenGet meas=<meas> prop=<prop>`

**CLI Abbreviation**    `perfgenet meas=<meas> prop=<prop>`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**    **handle**    Session identification.

**meas**    Value to identify the measurement; see “*Measurement Identifier*” on page 142.

**prop**    Property to be obtained; see “*bx\_perfgentype*” on page 236.

**Output Parameters**    **val**    Value of the queried property; see “*bx\_perfgentype*” on page 236.

**See also**    “*BestXPerfGenDefaultSet*” on page 144  
                  “*BestXPerfGenSet*” on page 146

## BestXPerfGenSet

**Call** `bx_errtype BestXPerfGenPropSet (`  
     `bx_handletyp     handle,`  
     `bx_int32          meas,`  
     `bx_perfgentype   prop,`  
     `bx_int32          val );`

**Description** Sets the value of a generic performance property of a performance measurement. These properties are used to:

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

**CLI Equivalent** `BestXPerfGenSet meas=<meas> prop=<prop> val=<val>`

**CLI Abbreviation** `perfgenset meas=<meas> prop=<prop> val=<val>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**meas** Value to identify the measurement; see “*Measurement Identifier*” on page 142.

**prop** Property to be set; see “*bx\_perfgentype*” on page 236.

**val** Value to which the property is set; see “*bx\_perfgentype*” on page 236.

**See also** “*BestXPerfGenDefaultSet*” on page 144  
 “*BestXPerfGenGet*” on page 145

## BestXPerfProg

**Call** `bx_errtype BestXPerfProg(  
 bx_handletype handle,  
 bx_int32 meas );`

**Description** Writes the properties that were previously set in the sequencer description table to the performance sequencer of the selected performance measurement.

This function also checks for syntax errors and logical inconsistencies within the programmed state machine. If any are found, the function returns an error.

**CLI Equivalent** `BestXPerfProg meas=<meas>`

**CLI Abbreviation** `perfprog meas=<meas>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**meas** Value to identify the measurement; see “*Measurement Identifier*” on page 142.

**See also** “*BestXPerfCondSet*” on page 141  
“*BestXPerfTranSet*” on page 149  
“*BestXPerfTranCondDefaultSet*” on page 148

## BestXPerfRun

**Call** `bx_errtype BestXPerfRun( bx_handletype handle );`

**Description** Starts all four performance sequencers simultaneously.

**CLI Equivalent** `BestXPerfRun`

**CLI Abbreviation** `perfrun`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPerfStop*” on page 150

## BestXPerfTranCondDefaultSet

**Call** `bx_errtype BestXPerfTranCondDefaultSet (
 bx_handletype handle,
 bx_int32 meas,
 bx_int32 offset );`

**Description** Sets all properties for a specific line and a specific measurement in the sequencer description table to default values.

For a description of properties and default values, refer to “*bx\_perfcondtype*” on page 235 and “*bx\_perftrantype*” on page 236.

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

**CLI Equivalent** `BestXPerfTranCondDefaultSet meas=<meas> offset=<offset>`

**CLI Abbreviation** `perftranconddefset meas=<meas> offset=<offset>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**meas** Value to identify the measurement; see “*Measurement Identifier*” on page 142.

**offset** Number (0 ... 127).

**See also** “*BestXPerfCondSet*” on page 141  
 “*BestXPerfTranSet*” on page 149  
 “*BestXTrigDefaultSet*” on page 125  
 “*BestXPerfProg*” on page 147

## BestXPerfTranSet

**Call** `bx_errtype BestXPerfTranSet (`  
     `bx_handletype handle,`  
     `bx_int32 meas,`  
     `bx_int32 offset,`  
     `bx_perftrantype prop,`  
     `bx_int32 val );`

**Description** Sets a numeric transition property (“state” or “next state”) for a specific performance measurement in the sequencer description table.

To write the property to the testcard, use the BestXPerfProg call.

**CLI Equivalent** `BestXPerfTranSet meas=<meas> offset=<offset> prop=<prop> val=<val>`

**CLI Abbreviation** `perftraset meas=<meas> offset=<offset> prop=<prop>`  
`val=<val>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**meas** Value to identify the measurement; see “*Measurement Identifier*” on page 142.

**offset** Number (0 ... 127).

**prop** Property to be set; see “*bx\_perftrantype*” on page 236.

**val** Value to which the property is set.

**See also** “*BestXPerfCondSet*” on page 141  
 “*BestXPerfTranCondDefaultSet*” on page 148  
 “*BestXPerfProg*” on page 147

## BestXPerfStop

**Call** `bx_errtype BestXPerfStop( bx_handletype handle );`

**Description** Stops all four performance sequencers simultaneously.

**CLI Equivalent** `BestXPerfStop`

**CLI Abbreviation** `perfstop`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPerfRun*” on page 148

## BestXPerfUpdate

**Call** `bx_errtype BestXPerfUpdate( bx_handletype handle );`

**Description** Copies the content of all performance counters into 64-bit wide shadow registers.

The counter values can be read with the `BestXPerfCtrRead` call.

**CLI Equivalent** `BestXPerfUpdate`

**CLI Abbreviation** `perfupdate`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPerfCtrRead*” on page 142

# Interface Control Functions

The PCI-X interface control functions are divided as follows:

Interface Control Functions	Action
Display Functions	Write to the 7-segment display.
Mailbox Functions	Communicate between testcard and test environment by means of the PCI-X interface.

## Display Functions

The following functions are used to control the 7-segment display:

Function	Action
<i>"BestXDisplayStringWrite" on page 152</i>	Writes a string to the display.
<i>"BestXDisplayWrite" on page 153</i>	Writes a value to the display.

## BestXDisplayStringWrite

**Call** `bx_errtype BestXDisplayStringWrite(  
 bx_handletype handle,  
 bx_charptrtype charptr );`

**Description** Writes a string to the 7-segment display.

Before using this function, ensure that the “user” mode is selected for the display. To verify this mode, read the board property `BX_BOARD_DISPLAY` by using the `BestXBoardGet` call. See “*bx\_boardtype*” on page 210.

**CLI Equivalent** `BestXDisplayStringWrite charptr=<“string”>`

**CLI Abbreviation** `displaystringwrite charptr=<“string”>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**charptr** String to be displayed. The maximum length is 4 characters.

**See also** “*BestXDisplayWrite*” on page 153  
“*BestXBoardGet*” on page 36



## BestXDisplayWrite

**Call** `bx_errtype BestXDisplayWrite(  
 bx_handletype handle,  
 bx_int32 value );`

**Description** Writes a value to the 7-segment display.

Before using this function, ensure that the “user” mode is selected for the display. To verify this mode, read the board property `BX_BOARD_DISPLAY` by using the `BestXBoardGet` call. See “*bx\_boardtype*” on page 210.

**CLI Equivalent** `BestXDisplayWrite value=<value>`

**CLI Abbreviation** `displaywrite val=<value>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**value** Number to be displayed. The maximum number is 0xffff.

**See also** “*BestXDisplayStringWrite*” on page 152  
“*BestXBoardGet*” on page 36

# Mailbox Functions

The following functions are used for communication between testcard and test environment by means of the PCI-X interface:

Function	Result
<i>"BestXMailboxReceiveRegRead" on page 154</i>	Reads data from the mailbox to the control PC.
<i>"BestXMailboxSendRegWrite" on page 155</i>	Sends data from the host to the mailbox.
<i>"BestXPCICfgMailboxReceiveRegRead" on page 156</i>	Reads data from the mailbox by means of PCI-X.
<i>"BestXPCICfgMailboxSendRegWrite" on page 157</i>	Sends data to the mailbox by means of PCI-X.

## BestXMailboxReceiveRegRead

**Call** `bx_errtype BestXMailboxReceiveRegRead(  
           bx_handletype  handle,  
           bx_int32      *value,  
           bx_int32      *status );`

**Description** This function is used on the host to receive a value from the mailbox by means of the control interface. The function returns the mailbox value and a status flag.

**CLI Equivalent** BestXMailboxReceiveRegRead

**CLI Abbreviation** mailboxreceiveregread

**Return Value** Error code; see *"bx\_errtype" on page 230*.

**Input Parameters** **handle** Session identification.

**Output Parameters** **value** Value read from the mailbox.

**status** Status flag:

- 0 – The mailbox was empty. The transfer failed.
- 1 – The transfer was successful.

**See also** *"BestXMailboxSendRegWrite" on page 155*  
*"BestXPCICfgMailboxReceiveRegRead" on page 156*

## BestXMailboxSendRegWrite

**Call** `bx_errtype BestXMailboxSendRegWrite(  
           bx_handletype  handle,  
           bx_int32      value,  
           bx_int32      *status );`

**Description** This function is used on the host to send a value to the mailbox by means of the control interface.

If the mailbox is occupied (by unread data, see the status flag below), the function will not write the value to the mailbox.

**CLI Equivalent** `BestXMailboxSendRegWrite value=<value>`

**CLI Abbreviation** `mailboxsendregwrite val=<value>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**value** Value to send to the mailbox.

**Output Parameters** **status** Status flag:

- 0 – Unread data was found in the mailbox. The transfer failed.
- 1 – The transfer was successful.

**See also** “*BestXMailboxReceiveRegRead*” on page 154  
 “*BestXPCICfgMailboxSendRegWrite*” on page 157

## BestXPCICfgMailboxReceiveRegRead

**Call** `bx_errtype BestXPCICfgMailboxReceiveRegRead (`  
     `bx_int32 devid,`  
     `bx_int32 *value,`  
     `bx_int32 *status );`

**Description** This function is used on the system under test to read a value from the mailbox by means of the PCI-X bus. The function returns the mailbox value and the status flag.

To identify the testcard within the PCI-X system, first call `BestXDevIdentifierGet`. The device identifier returned by this function can be used directly.

**CLI Equivalent** No CLI equivalent.

**CLI Abbreviation** No CLI abbreviation.

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **devid** Device identifier of the testcard as returned by the `BestXDevIdentifierGet` call.

**Output Parameters** **value** Value read from the mailbox.

**status** Status flag:

- 0 – The mailbox was empty. The transfer failed.
- 1 – The transfer was successful.

**See also** “*BestXMailboxReceiveRegRead*” on page 154  
 “*BestXPCICfgMailboxSendRegWrite*” on page 157

## BestXPCICfgMailboxSendRegWrite

**Call** `bx_errtype BestXPCICfgMailboxSendRegWrite(  
     bx_int32 devid,  
     bx_int32 value,  
     bx_int32 *status );`

**Description** This function is used on the system under test to write a value to the mailbox by means of the PCI-X bus.

To identify the testcard within the PCI-X system, first call `BestXDevIdentifierGet`. The device identifier returned by this function can be used directly.

If the mailbox is occupied (by unread data), the function will not write the value to the mailbox.

**CLI Equivalent** No CLI equivalent.

**CLI Abbreviation** No CLI abbreviation.

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **devid** Device identifier of the testcard as returned by the `BestXDevIdentifierGet` call.

**value** Value to send to the mailbox.

**Output Parameters** **status** Status flag:

- 0 – Unread data was found in the mailbox. The transfer failed.
- 1 – The transfer was successful.

**See also** “*BestXPCICfgMailboxReceiveRegRead*” on page 156  
 “*BestXMailboxSendRegWrite*” on page 155



# 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
<i>"BestXPprInit" on page 161</i>	Initializes the PPR software.
<i>"BestXPprProg" on page 161</i>	Writes all current PPR settings to the testcard.
<i>"BestXPprDelete" on page 160</i>	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 230*.

**Input Parameters** **handle** Session identification.

**See also** *"BestXPprInit" on page 161*



## 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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPprDelete*” on page 160

## 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 63).

You can define which parts of the exerciser are programmed by setting the `BXPPR_GEN_USE_xx` generic properties (see “*BestXPprGenSet*” on page 164).

**CLI Equivalent** `BestXPprProg`

**CLI Abbreviation** `pprprog`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** -

# PPR Generic Functions

The following functions are used to prepare for PPR programming:

Function	Action
<i>"BestXPprGenDefaultSet" on page 162</i>	Sets all generic PPR properties to default values.
<i>"BestXPprGenSet" on page 164</i>	Sets a generic PPR property.
<i>"BestXPprGenGet" on page 163</i>	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 263*.

**CLI Equivalent** `BestXPprGenDefaultSet`

**CLI Abbreviation** `pprgendefset`

**Return Value** Error code; see *"bx\_errtype" on page 230*.

**Input Parameters** **handle** Session identification.

**See also** *"BestXPprGenSet" on page 164*  
*"BestXPprGenGet" on page 163*  
*"BestXPprProg" on page 161*

## 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 230.

**Input Parameters**    **handle**    Session identification.

**genprop**    Property to be obtained; see “*bxppr\_gentype*” on page 263.

**Output Parameters**    **pValue**    Value of the property; see “*bxppr\_gentype*” on page 263.

**See also**    “*BestXPprGenSet*” on page 164  
              “*BestXPprGenDefaultSet*” on page 162

## 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 230.

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

**prop**    Property to be set; see “*bxppr\_gentype*” on page 263.

**value**    Value of the property to be set; see “*bxppr\_gentype*” on page 263.

**See also**    “*BestXPprGenDefaultSet*” on page 162  
                   “*BestXPprGenGet*” on page 163  
                   “*BestXPprProg*” on page 161

# PPR Requester-Initiator Functions

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

Function	Action
<i>"BestXPprRIDefaultSet" on page 177</i>	Sets all PPR requester-initiator properties to default values.
<i>"BestXPprRIBlkPermDefaultSet" on page 175</i>	Sets all requester-initiator block permutation properties to default values.
<i>"BestXPprRIBlkPermSet" on page 176</i>	Sets a requester-initiator block permutation property.
<i>"BestXPprRIBlkPermGet" on page 175</i>	Gets a requester-initiator block permutation property.
<i>"BestXPprRIBlkListDefaultSet" on page 172</i>	Sets all list values from a requester-initiator block property to default values.
<i>"BestXPprRIBlkListSet" on page 174</i>	Sets the list values from a requester-initiator block property.
<i>"BestXPprRIBlkListGet" on page 173</i>	Gets the list values from a requester-initiator block property.
<i>"BestXPprRIBlkResultGet" on page 177</i>	Gets the permutation result from a requester-initiator result property.
<i>"BestXPprRIBlkGapGet" on page 178</i>	Gets a gap property for a specific gap.
<i>"BestXPprRIBehPermDefaultSet" on page 169</i>	Sets all requester-initiator behavior permutation properties to default values.
<i>"BestXPprRIBehPermSet" on page 170</i>	Sets a requester-initiator behavior permutation property.
<i>"BestXPprRIBehPermGet" on page 169</i>	Gets a requester-initiator behavior permutation property.
<i>"BestXPprRIBehListDefaultSet" on page 166</i>	Sets the value list of a requester-initiator behavior property to default values.
<i>"BestXPprRIBehListSet" on page 168</i>	Sets the value list of a requester-initiator behavior property.
<i>"BestXPprRIBehListGet" on page 167</i>	Gets the value list of a requester-initiator behavior property.
<i>"BestXPprRIBehResultGet" on page 171</i>	Gets the permutation result of a PPR requester-initiator behavior result property.

## BestXPprRIBehListDefaultSet

**Call** `bx_errtype BestXPprRIBehListDefaultSet (`  
     `bx_handletype handle,`  
     `bx_ribehetype 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\_ribehetype*” on page 239.

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

**CLI Equivalent** `BestXPprRIBehListDefaultSet behavior=<behavior>`

**CLI Abbreviation** `ppribehlistdefaultset beh=<behavior>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**behavior** Behavior property to be set; see “*bx\_ribehetype*” on page 239.

**See also** “*BestXPprRIBehListSet*” on page 168  
 “*BestXPprRIBehListGet*” on page 167  
 “*BestXPprProg*” on page 161

## BestXPprRIBehListGet

**Call**    `bx_errtype BestXPprRIBehListGet (  
          bx_handletype    handle,  
          bx_ribetype      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**    `ppribehlistget beh=<behavior>`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**    **handle**    Session identification.

**behavior**    Behavior property to be obtained; see “*bx\_ribetype*” on page 239.

**Output Parameters**    **pList**    Value list of the behavior property. For the syntax of the list, see “*bxppr\_listtype*” on page 264.

**See also**    “*BestXPprRIBehListDefaultSet*” on page 166  
              “*BestXPprRIBehListSet*” on page 168

## BestXPprRIBehListSet

**Call** `bx_errtype BestXPprRIBehListSet (`  
     `bx_handletype     handle,`  
     `bx_ribetype     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** `ppribehlistset beh=<behavior> list=<list>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**behavior** Behavior property to be set; see “*bx\_ribetype*” on page 239.

**pList** Value list of the behavior property to be set. For the syntax of the list, see “*bxppr\_listtype*” on page 264.

**See also** “*BestXPprRIBehListDefaultSet*” on page 166  
           “*BestXPprRIBehListGet*” on page 167  
           “*BestXPprProg*” on page 161



## 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 261.

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

**CLI Equivalent** `BestXPprRIBehPermDefaultSet`

**CLI Abbreviation** `ppribehpermdefaultset`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPprRIBehPermSet*” on page 170  
 “*BestXPprRIBehPermGet*” on page 169  
 “*BestXPprProg*” on page 161

## 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** `ppribehpermget prop=<permprop>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**permprop** Behavior property to be obtained; see “*bxppr\_behpermtype*” on page 261.

**Output Parameters** **pValue** Value of the behavior property; see “*bxppr\_behpermtype*” on page 261.

**See also** “*BestXPprRIBehPermDefaultSet*” on page 169  
 “*BestXPprRIBehPermSet*” on page 170

## 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 230.

**Input Parameters**    **handle**    Session identification.

**permprop**    Behavior property to be set; see “*bxppr\_behpermtype*” on page 261.

**value**    Value of the behavior property to be set; see “*bxppr\_behpermtype*” on page 261.

**See also**    “*BestXPprRIBehPermDefaultSet*” on page 169  
              “*BestXPprRIBehPermGet*” on page 169  
              “*BestXPprProg*” on page 161

## 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**    `ppribehresultget prop=<resultprop>`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**    **handle**    Session identification.

**resultprop**    Behavior result property to be obtained; see “*bxppr\_behresulttype*” on page 262.

**Output Parameters**    **pValue**    Value of the behavior result property; “*bxppr\_behresulttype*” on page 262.

**See also**    “*BestXPprRIBlkResultGet*” on page 177

## 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** `pprribblklistdefaultset prop=<blkprop>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**blkprop** Block property from which the list values are to be set; see “*bxppr\_riblktype*” on page 270.

**See also** “*BestXPprRIBlkListSet*” on page 174  
           “*BestXPprRIBlkListGet*” on page 173  
           “*BestXPprProg*” on page 161

## 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**   `ppriblklistget prop=<blkprop>`

**Return Value**     Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**   **handle**     Session identification.

**blkprop**     Property from which the list values are to be obtained; see “*bxppr\_riblktype*” on page 270.

**Output Parameters**   **pList**     List values of the property. For the syntax of the list, see “*bxppr\_listtype*” on page 264.

**See also**     “*BestXPprRIBlkListDefaultSet*” on page 172  
                  “*BestXPprRIBlkListSet*” on page 174

## 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** `pprribklistset prop=<blkprop> list=<list>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**blkprop** Block property from which the list values are to be set; see “*bxppr\_riblktype*” on page 270.

**pList** List values of the property to be set. For the syntax of the list, see “*bxppr\_listtype*” on page 264.

**See also** “*BestXPprRIBlkListDefaultSet*” on page 172  
           “*BestXPprRIBlkListGet*” on page 173  
           “*BestXPprProg*” on page 161

## 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 268.

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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPprRIBlkPermSet*” on page 176  
 “*BestXPprRIBlkPermGet*” on page 175  
 “*BestXPprProg*” on page 161

## 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 230.

**Input Parameters** **handle** Session identification.

**permprop** Block permutation property to be obtained; see “*bxppr\_riblkpermtype*” on page 268.

**Output Parameters** **pValue** Value of the property; see “*bxppr\_riblkpermtype*” on page 268.

**See also** “*BestXPprRIBlkPermDefaultSet*” on page 175  
 “*BestXPprRIBlkPermSet*” on page 176

## BestXPprRIBlkPermSet

**Call** `bx_errtype BestXPprRIBlkPermSet (`  
     `bx_handletype           handle,`  
     `bxppr_riblkpermtypes   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 230.

**Input Parameters** **handle** Session identification.

**permprop** Block permutation property to be set; see  
 “*bxppr\_riblkpermtypes*” on page 268.

**value** Value of the property to be set; see “*bxppr\_riblkpermtypes*” on  
 page 268.

**See also** “*BestXPprRIBlkPermDefaultSet*” on page 175  
 “*BestXPprRIBlkPermGet*” on page 175  
 “*BestXPprProg*” on page 161



## 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** `ppriblkresultget prop=<resultprop>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**resultprop** Block result property from which the results are to be obtained; see “*bxppr\_riblkresulttype*” on page 269.

**Output Parameters** **pValue** Result of the property; see “*bxppr\_riblkresulttype*” on page 269.

**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** `ppridefaultset`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPprRIBlkPermDefaultSet*” on page 175  
 “*BestXPprRIDefaultSet*” on page 177  
 “*BestXPprRIBehPermDefaultSet*” on page 169  
 “*BestXPprRIBehListDefaultSet*” on page 166  
 “*BestXPprProg*” on page 161

## BestXPprRIBlkGapGet

**Call**    `bx_errtype BestXPprRIBlkGapGet (`  
                  `bx_handletype            handle,`  
                  `bx_int32                            gapnum,`  
                  `bxppr_ribkgaptype            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**    `pprribkgapget gapnum=<gapnum> type=<gapprop>`

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**    **handle**    Session identification.

**gapnum**    Number of the gap that is referenced.

**gapprop**    Gap property to be obtained; see “*bxppr\_ribkgaptype*” on page 266.

**Output Parameters**    **pValue**    Value of the gap property; see “*bxppr\_ribkgaptype*” on page 266.

**See also**    –

# PPR Requester-Target Functions

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

Function	Action
<i>"BestXPprRTDefaultSet" on page 186</i>	Sets all PPR requester-target properties to default values.
<i>"BestXPprRTBehPermDefaultSet" on page 183</i>	Sets all PPR requester-target behavior properties to default values.
<i>"BestXPprRTBehPermSet" on page 185</i>	Sets a PPR requester-target behavior property.
<i>"BestXPprRTBehPermGet" on page 184</i>	Gets a PPR requester-target behavior property.
<i>"BestXPprRTBehListDefaultSet" on page 180</i>	Sets all list values of PPR requester-target behavior properties to default values.
<i>"BestXPprRTBehListSet" on page 182</i>	Sets the list values of a PPR requester-target behavior property.
<i>"BestXPprRTBehListGet" on page 181</i>	Gets the list values of a PPR requester-target behavior property.
<i>"BestXPprRTBehResultGet" on page 186</i>	Gets the permutation result of a PPR requester-target behavior result property.

## BestXPprRTBehListDefaultSet

**Call** `bx_errtype BestXPprRTBehListDefaultSet (`  
     `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 245.

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 230.

**Input Parameters** **handle** Session identification.

**behavior** Behavior property to be set; see “*bx\_rtbehtype*” on page 245.

**See also** “*BestXPprRTBehListSet*” on page 182  
 “*BestXPprRTBehListGet*” on page 181  
 “*BestXPprProg*” on page 161

## 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 230.

**Input Parameters** **handle** Session identification.

**behavior** Behavior property to be obtained; see “*bx\_rtbehtype*” on page 245.

**Output Parameters** **pList** Value list of the behavior property. For the syntax of the list, see “*bxppr\_listtype*” on page 264.

**See also** “*BestXPprRTBehListDefaultSet*” on page 180  
 “*BestXPprRTBehListSet*” on page 182

## 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 230.

**Input Parameters** **handle** Session identification.

**behavior** Behavior property to be set; see “*bx\_rtbehtype*” on page 245.

**pList** Value list of the behavior property to be set. For the syntax of the list, see “*bxppr\_listtype*” on page 264.

**See also** “*BestXPprRTBehListDefaultSet*” on page 180  
           “*BestXPprRTBehListGet*” on page 181  
           “*BestXPprProg*” on page 161

## 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 261.

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

**CLI Equivalent**    BestXPprRIBehPermDefaultSet

**CLI Abbreviation**    ppribehpermdefaultset

**Return Value**    Error code; see “*bx\_errtype*” on page 230.

**Input Parameters**    **handle**    Session identification.

**See also**    “*BestXPprRTBehPermSet*” on page 185  
              “*BestXPprRTBehPermGet*” on page 184  
              “*BestXPprProg*” on page 161

## 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** `pprtrtbehpermget prop=<permprop>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**permprop** Behavior permutation property to be obtained; see “*bxppr\_behpermtype*” on page 261.

**Output Parameters** **pValue** Value of the behavior permutation property; see “*bxppr\_behpermtype*” on page 261.

**See also** “*BestXPprRTBehPermDefaultSet*” on page 183  
           “*BestXPprRTBehPermSet*” on page 185



## 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** `pprtrtbehpermset prop=<permprop> value=<value>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**permprop** Behavior permutation property to be set; see  
 “*bxppr\_behpermtype*” on page 261.

**value** Value of the behavior permutation property to be set; see  
 “*bxppr\_behpermtype*” on page 261.

**See also** “*BestXPprRTBehPermDefaultSet*” on page 183  
 “*BestXPprRTBehPermGet*” on page 184  
 “*BestXPprProg*” on page 161

## 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** `pprtrtbehresultget prop=<resultprop>`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**resultprop** Behavior result property to be obtained; see “*bxppr\_behresulttype*” on page 262.

**Output Parameters** **pValue** Value of the behavior result property; “*bxppr\_behresulttype*” on page 262.

**See also** “*BestXPprRIBlkResultGet*” on page 177

## 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** `pprtrtdefaultset`

**Return Value** Error code; see “*bx\_errtype*” on page 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPprRTBehPermDefaultSet*” on page 183

“*BestXPprRIBehPermDefaultSet*” on page 169

“*BestXPprRTBehListDefaultSet*” on page 180

“*BestXPprProg*” on page 161

# PPR Completer-Target Functions

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

Function	Action
<i>"BestXPprCTDefaultSet" on page 193</i>	Sets all PPR completer-target properties to default values.
<i>"BestXPprCTBehPermDefaultSet" on page 191</i>	Sets all completer-target behavior permutation properties to default values.
<i>"BestXPprCTBehPermSet" on page 192</i>	Sets a completer-target behavior permutation property.
<i>"BestXPprCTBehPermGet" on page 191</i>	Gets a completer-target behavior permutation property.
<i>"BestXPprCTBehListDefaultSet" on page 188</i>	Sets the value list of a completer-target behavior property to default values.
<i>"BestXPprCTBehListSet" on page 190</i>	Sets the value list of a completer-target behavior property.
<i>"BestXPprCTBehListGet" on page 189</i>	Gets the value list of a completer-target behavior property.
<i>"BestXPprCTBehResultGet" on page 193</i>	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 215.

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 230.

**Input Parameters** **handle** Session identification.

**behavior** Behavior property to be set; see “*bx\_ctbehtype*” on page 215.

**See also** “*BestXPprCTBehListSet*” on page 190  
 “*BestXPprCTBehListGet*” on page 189  
 “*BestXPprProg*” on page 161

## 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 230.

**Input Parameters**    **handle**    Session identification.

**behavior**    Behavior property to be obtained; see “*bx\_ctbehtype*” on page 215.

**Output Parameters**    **pList**    Value list of the behavior property. For the syntax of the list, see “*bxppr\_listtype*” on page 264.

**See also**    “*BestXPprCTBehListDefaultSet*” on page 188  
              “*BestXPprCTBehListSet*” on page 190

## 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 230.

**Input Parameters** **handle** Session identification.

**behavior** Behavior property to be set; see “*bx\_ctbehtype*” on page 215.

**pList** Value list of the behavior property to be set. For the syntax of the list, see “*bxppr\_listtype*” on page 264.

**See also** “*BestXPprCTBehListDefaultSet*” on page 188  
           “*BestXPprCTBehListGet*” on page 189  
           “*BestXPprProg*” on page 161

## 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 261.

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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPprCTBehPermSet*” on page 192  
 “*BestXPprCTBehPermGet*” on page 191  
 “*BestXPprProg*” on page 161

## 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 230.

**Input Parameters** **handle** Session identification.

**permprop** Behavior property to be obtained; see “*bxppr\_behpermtype*” on page 261.

**Output Parameters** **pValue** Value of the behavior permutation property; see “*bxppr\_behpermtype*” on page 261.

**See also** “*BestXPprCTBehPermDefaultSet*” on page 191  
 “*BestXPprCTBehPermSet*” on page 192

## 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 230.

**Input Parameters** **handle** Session identification.

**permprop** Behavior property to be set; see “*bxppr\_behpermtype*” on page 261.

**value** Value of the behavior property to be set; see “*bxppr\_behpermtype*” on page 261.

**See also** “*BestXPprCTBehPermDefaultSet*” on page 191  
 “*BestXPprCTBehPermGet*” on page 191  
 “*BestXPprProg*” on page 161



## 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 230.

**Input Parameters** **handle** Session identification.

**resultprop** Behavior result property to be obtained; see “*bxppr\_behresulttype*” on page 262.

**Output Parameters** **pValue** Value of the behavior result property; “*bxppr\_behresulttype*” on page 262.

**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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPprCTBehPermDefaultSet*” on page 191  
 “*BestXPprCTBehListDefaultSet*” on page 188  
 “*BestXPprProg*” on page 161

# PPR Completer-Initiator Functions

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

Function	Action
<i>"BestXPprCIDefaultSet" on page 200</i>	Sets all PPR completer-initiator properties to default values.
<i>"BestXPprCIBehPermDefaultSet" on page 198</i>	Sets all completer-initiator behavior permutation properties to default values.
<i>"BestXPprCIBehPermSet" on page 199</i>	Sets a completer-initiator behavior permutation property.
<i>"BestXPprCIBehPermGet" on page 198</i>	Gets a completer-initiator behavior permutation property.
<i>"BestXPprCIBehListDefaultSet" on page 195</i>	Sets the value list of a completer-initiator behavior property to default values.
<i>"BestXPprCIBehListSet" on page 197</i>	Sets the value list of a completer-initiator behavior property.
<i>"BestXPprCIBehListGet" on page 196</i>	Gets the value list of a completer-initiator behavior property.
<i>"BestXPprCIBehResultGet" on page 200</i>	Gets the permutation result of a PPR completer-initiator behavior result property.

## BestXPprCIBehListDefaultSet

**Call** `bx_errtype BestXPprCIBehListDefaultSet (  
    bx_handletype handle,  
    bx_cibehype 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\_cibehype*” on page 213.

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 230.

**Input Parameters** **handle** Session identification.

**behavior** Behavior property to be set; see “*bx\_cibehype*” on page 213.

**See also** “*BestXPprCIBehListSet*” on page 197  
“*BestXPprCIBehListGet*” on page 196  
“*BestXPprProg*” on page 161

## BestXPprCIBehListGet

**Call**    `bx_errtype BestXPprCIBehListGet (  
                   bx_handletype    handle,  
                   bx_cibehotype    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 230.

**Input Parameters**    **handle**    Session identification.

**behavior**    Behavior property to be obtained; see “*bx\_cibehotype*” on page 213.

**Output Parameters**    **pList**    Value list of the behavior property. For the syntax of the list, see “*bxppr\_listtype*” on page 264.

**See also**    “*BestXPprCIBehListDefaultSet*” on page 195  
                   “*BestXPprCIBehListSet*” on page 197

## BestXPprCIBehListSet

**Call**    `bx_errtype BestXPprCIBehListSet (`  
              `bx_handletype     handle,`  
              `bx_cibehotype     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 230.

**Input Parameters**    **handle**    Session identification.

**behavior**    Behavior property to be set; see “*bx\_cibehotype*” on page 213.

**pList**    Value list of the behavior property to be set. For the syntax of the list, see “*bxppr\_listtype*” on page 264.

**See also**    “*BestXPprCIBehListDefaultSet*” on page 195  
              “*BestXPprCIBehListGet*” on page 196  
              “*BestXPprProg*” on page 161

## 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 261.

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 230.

**Input Parameters** **handle** Session identification.

**See also** “BestXPprCIBehPermSet” on page 199  
 “BestXPprCIBehPermGet” on page 198  
 “BestXPprProg” on page 161

## 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 230.

**Input Parameters** **handle** Session identification.

**permprop** Behavior property to be obtained; see “*bxppr\_behpermtype*” on page 261.

**Output Parameters** **pValue** Value of the behavior property; see “*bxppr\_behpermtype*” on page 261.

**See also** “BestXPprCIBehPermDefaultSet” on page 198  
 “BestXPprCIBehPermSet” on page 199

## 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 230.

**Input Parameters** **handle** Session identification.

**permprop** Behavior property to be set; see “*bxppr\_behpermtype*” on page 261.

**value** Value of the behavior property to be set; see “*bxppr\_behpermtype*” on page 261.

**See also** “*BestXPprCIBehPermDefaultSet*” on page 198  
 “*BestXPprCIBehPermGet*” on page 198  
 “*BestXPprProg*” on page 161

## 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 230.

**Input Parameters** **handle** Session identification.

**resultprop** Behavior result property to be obtained; see “*bxppr\_behresulttype*” on page 262.

**Output Parameters** **pValue** Value of the behavior result property; “*bxppr\_behresulttype*” on page 262.

**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 230.

**Input Parameters** **handle** Session identification.

**See also** “*BestXPprCIBehPermDefaultSet*” on page 198  
 “*BestXPprCIBehListDefaultSet*” on page 195  
 “*BestXPprProg*” on page 161



# PPR Reporting Functions

The following functions are used for PPR reporting:

Function	Action
<i>"BestXPprReportWrite" on page 201</i>	Generates a PPR report.
<i>"BestXPprReportFile" on page 202</i>	Writes the report to a file.
<i>"BestXPprReportSet" on page 203</i>	Sets report properties.
<i>"BestXPprReportGet" on page 202</i>	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 230*.

**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 202*  
*"BestXPprReportSet" on page 203*  
*"BestXPprReportGet" on page 202*

## 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 230.

**Input Parameters** **handle** Session identification.

**filename** Name of the file to which the report is to be written.

**See also** “*BestXPprReportWrite*” on page 201  
 “*BestXPprReportSet*” on page 203  
 “*BestXPprReportGet*” on page 202

## 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 230.

**Input Parameters** **handle** Session identification.

**reportprop** Report property to be obtained; see “*bxppr\_reporttype*” on page 265.

**Output Parameters** **pValue** Value of the report property; see “*bxppr\_reporttype*” on page 265.

**See also** “*BestXPprReportSet*” on page 203

## 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 230.

**Input Parameters**    **handle**    Session identification.

**reportprop**    Report property to be set; see “*bxppr\_reporttype*” on page 265.

**value**    Value of the report property to be set; see “*bxppr\_reporttype*” on page 265.

**See also**    “*BestXPprReportGet*” on page 202



# 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
<i>"BestXLastErrorGet" on page 207</i>	Returns the last error.
<i>"BestXLastErrorStringGet" on page 207</i>	Returns the error string of the last error.
<i>"BestXErrorStringGet" on page 206</i>	Returns the error string of an error.

## BestXErrorStringGet

**Call**     `bx_charptrtype BestXErrorStringGet (`  
                  `bx_errtype                    error );`

**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**     **error**     Error code; see “*bx\_errtype*” on page 230.

**See also**     “*BestXLastErrorGet*” on page 207  
                  “*BestXLastErrorStringGet*” on page 207

## 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 230.

**Input Parameters** **handle** Session identification.

**Output Parameters** **error** Error code; see “*bx\_errtype*” on page 230. If no error occurred, BX\_E\_OK is returned.

**See also** “*BestXLastErrorStringGet*” on page 207  
“*BestXErrorStringGet*” on page 206

## 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 207  
“*BestXErrorStringGet*” on page 206





# 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.

prop (CLI Abbreviation)	val	Description
BX_BOARD_PCIXCAP (pcixcap)	<p>Available only for E2930 and E2923 testcards.</p> <p>System configuration: Specifies the capabilities the card is to signal to the system BIOS through the PCIXCAP and MODE2 pins. The state is set with the help of the board's relays. See also BX_STAT_M66EN below and BX_STAT_RESETCODE ( "<i>bx_statustype</i>" on page 253).</p> <p>The states of the relays cannot be detected by the software. Therefore, the value of this property is initially set to unknown (BX_BOARD_PCIXCAP_UNKNOWN).</p>	
	BX_BOARD_PCIXCAP_PCI	Testcard identifies itself as a conventional PCI device.
	BX_BOARD_PCIXCAP_PCIX66 (default)	Testcard identifies itself as a PCI-X device supporting up to 66 MHz.
	BX_BOARD_PCIXCAP_PCIX133	Testcard identifies itself as a PCI-X device supporting up to 133 MHz.
	BX_BOARD_PCIXCAP_PCIX266	Testcard identifies itself as a PCI-X device supporting up to 266 MHz.
	BX_BOARD_PCIXCAP_PCIX533	Testcard identifies itself as a PCI-X device supporting up to 533 MHz.
BX_BOARD_M66EN (m66en)	<p>Available only for E2930 and E2923 testcards.</p> <p>System configuration: Pulls the PCI M66EN pin to ground (33 MHz) or leaves it open (66 MHz capable) with the help of the board's relays. See also BX_BOARD_PCIXCAP above and BX_STAT_RESETCODE ( "<i>bx_statustype</i>" on page 253).</p> <p>The states of the relays cannot be detected by the software. Therefore, the value of this property is initially set to unknown (BX_BOARD_M66EN_UNKNOWN).</p>	
	BX_BOARD_M66EN_GND	<p>Pulls the M66EN pin to ground.</p> <p>Conventional PCI device frequency capability: 33 MHz.</p>
	BX_BOARD_M66EN_OPEN	<p>M66EN not connected.</p> <p>Conventional PCI device frequency capability: 66 MHz.</p>
	BX_BOARD_M66EN_UNKNOWN (default)	Capabilities unknown.

prop (CLI Abbreviation)	val	Description
BX_BOARD_RESPECTBIOS (respectbios)	Defines the behavior of the testcard at power up or reset with regard to the content of the configuration space.	
	default: 1	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.
	0	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.
BX_BOARD_BUSCLOCK (busclock)	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.	
	default: BX_BOARD_BUSCLOCK_DEF	Standard synchronization method. This will work in almost 100% of all cases.
	BX_BOARD_BUSCLOCK_SLOW	The synchronization of the internal ASIC register is forced. This is <i>only</i> to be used with extremely slow bus speeds.
BX_BOARD_DISPLAY (display)	Display properties specify how the 7-segment display on the testcard is to be used.	
	BX_BOARD_DISPLAY_USER	You can use the 7-segment display to display any arbitrary string or number (by using BestXDisplayWrite or BestXDisplayStringWrite).
	default: BX_BOARD_DISPLAY_DEF	The testcard displays bus frequency and will display a protocol error if one occurs.
	BX_BOARD_DISPLAY_PERF	The display shows constantly the ratio of the two counters of the first performance measure. This way, a constant utilization or efficiency display (or whatever is programmed in the first performance measure) can be used.
BX_BOARD_TRIGIO0_MODE (trigio0mode), BX_BOARD_TRIGIO1_MODE (trigio1mode), BX_BOARD_TRIGIO2_MODE (trigio2mode), BX_BOARD_TRIGIO3_MODE (trigio3mode)	Sets the output mode of trigger out line 0 (1, 2, 3). Input is always possible.	
	default: BX_BOARD_TRIGIO_MODE_INONLY	Disables the output.
	BX_BOARD_TRIGIO_MODE_TOTEMPOLE	Sets the output to totem pole.
	BX_BOARD_TRIGIO_MODE_OPENDRAIN	Sets the output to open-drain.

prop (CLI Abbreviation)	val	Description
BX_BOARD_TRIGIO0_OUT (trigio0out),	Sets the output value of trigger-out line 0 (1, 2, 3). The property maps the trigger-out signals to internal signals. Trigger IO signals are low active (in open-drain and totem pole mode).	
BX_BOARD_TRIGIO1_OUT (trigio1out),	default for pin 0: BX_BOARD_TRIGIO_OUT_TRACETRIG	The trace memory trigger signal is used as output signal.
BX_BOARD_TRIGIO2_OUT (trigio2out),	default for pin 1: BX_BOARD_TRIGIO_OUT_PROTERR	The signal is asserted as soon as a protocol error occurs.
BX_BOARD_TRIGIO3_OUT (trigio3out)	default for pin 2: BX_BOARD_TRIGIO_OUT_DATACOMP	This setting is always asserted if a data compare error occurs. It is also asserted if a target abort or a split error message occurs (with switched-on data compare).
	default for pin 3: BX_BOARD_TRIGIO_OUT_TRIGSOURCE	The setting of BX_EGEN_TRIG_SOURCE is used as output (see <i>"bx_egentype" on page 222</i> ).

# bx\_cibehtype

prop (CLI Abbreviation)	Property Description	
	val	Value Description
BX_CIBEH_QUEUE (queue)	<p>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.</p> <p><b>Caution:</b> If not all queues are selected, completion of requests parked in the queue that has not been selected is stalled forever.</p> <p>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.</p>	
	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	<p>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.</p> <p><b>Caution:</b> Be very careful using this option, because requests in other queues will not be completed if the selected queue does not receive a request.</p> <p>For programming a PCI-X-compliant behavior, do <b>not</b> use these options. A device is not supposed to make a completion dependent on the receipt of another transaction.</p>
BX_CIBEH_ERRMESSAGE (errmessage)	<p>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.</p>	
	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 <b>not</b> 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 “ <i>bx_ctsplittype</i> ” on page 218), 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_SUBSEQ_TABORT	Available only for E2930 and E2923 testcards.  Signals a target abort.
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 <b>not</b> 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 “ <i>Decoder Programming Functions</i> ” on page 98).	
	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). Not available for E2930B testcards.	
	0 ... 0xffffffff default: 0	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.  Not available for E2930B testcards.
BX_CTSPLIT_ADDRVAL_HI (addrvalhi), BX_CTSPLIT_ADDRVAL_LO (addrvallo)	0 ... 0xffffffff default: 0	See BX_CTSPLIT_ADDRMASK_xx.  Not available for E2930B testcards.
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_ALL	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.	
defaults:	BX_CTSPLIT_DEC_1	Address range number 1 (BAR 0 and BAR 1).
	BX_CTSPLIT_DEC_2	Address range number 2 (BAR 2 and BAR 3).
Split decoder 0: BX_CTSPLIT_DEC_1	BX_CTSPLIT_DEC_3	Address range number 3 (BAR 4 and BAR 5).
	BX_CTSPLIT_DEC_ANY	Any address range.
Split decoder 1: BX_CTSPLIT_DEC_2	BX_CTSPLIT_DEC_NONE	No address range.
	BX_CTSPLIT_BAR_ANY	Any base address register.
Split decoder 2: BX_CTSPLIT_DEC_3	BX_CTSPLIT_BAR_0	Base address register 0 ... 5.
Split decoder 3: BX_CTSPLIT_DEC_NONE	BX_CTSPLIT_BAR_1	<b>Note:</b> 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.
	BX_CTSPLIT_BAR_2	
	BX_CTSPLIT_BAR_3	
	BX_CTSPLIT_BAR_4	
	BX_CTSPLIT_BAR_5	

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.  <b>Caution:</b> 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.  <b>Caution:</b> 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_DECP_RESOURCE_CFG	Decoder mapped to configuration space.  <b>Caution:</b> Works only with the IO decoder and a fixed behavior.

## 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_WRPAP, BX_EGEN_ERR_WRPAP64, 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_SUBPHASE (PCI-X Mode 2 only)	Only available for E2930 and E2923 testcards.  When the testcard is in PCI-X Mode 2 (DDR/QDR), this property allows to additionally specify the data subphase where the error is generated.	
	0 ... 15	The individual bits indicate in which subphases the errors should be inserted:  Bit 0 = 1: error inserted in subphase 1 Bit 1 = 1: error inserted in subphase 2 Bit 2 = 1: error inserted in subphase 3 Bit 3 = 1: error inserted in subphase 4  Default: 0 This default is handled by the HW exactly like BX_EGEN_ERR_SUBPHASE=1; the error is generated in subphase 1 (if BX_EGEN_ERR_ECC >0).
BX_EGEN_ERR_WRPAP (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_WRPAP.	
	default: 0	Parity is generated correctly.
	1	Parity is inverted.
BX_EGEN_ERR_WRPAP64 (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_WRPAP64.	
	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.

prop (CLI Abbreviation)	Value Description	
	val	Value Description
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.
BX_EGEN_ERR_ECC (errecc)	Available only for E2930 and E2923 testcards.  Controls injection of ECC errors.	
	default: 0	No error injected.
	1	Single bit error injected.
	2	Double bit error injected.



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_INT_RDONE (intrdone)	Available only for E2930 and E2923 testcards.  Specifies interrupt(s) to be asserted whenever the requester has finished (including all split completions). To specify more than one interrupt, the following values can be ORed.	
	default: 0	No interrupt will be asserted.
	BX_INTA	Interrupt A will be asserted.
	BX_INTB	Interrupt B will be asserted.
	BX_INTC	Interrupt C will be asserted.
	BX_INTD	Interrupt D will be asserted.

prop (CLI Abbreviation)	Value Description	
	val	Value Description
BX_EGEN_TRIG_SOURCE (trigsouce)	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).
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	

prop (CLI Abbreviation)	Value Description	
	val	Value Description
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.  <b>Caution:</b> 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.  <b>Caution:</b> 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 <i>&lt;parameter&gt;</i> must be aligned to <i>&lt;alignment&gt;</i> . Its value is <i>&lt;value&gt;</i> .	
BX_E_BAD_DECODER_NUMBER	No valid decoder number.	
BX_E_BAD_FILE_FORMAT	Bad format for file <i>&lt;filename&gt;</i> .	
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 <i>&lt;port&gt;</i> 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	<i>&lt;port&gt;</i> 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

**NOTE** All rules are valid for both PCI-X Mode 1 and Mode 2 unless otherwise noted.

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

Bit Pos.	rule	CLI Abbr.	Description	Reference
14	BX_OBSRULE_IRDY_1	irdy_1	Initiator Wait States are not permitted.	PCI-X Spec 1.10.2. Rule 3b
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 (PCI-X Mode 1 only)	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 (PCI-X Mode 1 only)	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 (PCI-X Mode 1 only)	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 (PCI-X Mode 1 only)	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 3.7.4
33	BX_OBSRULE_PAR_1 (PCI-X Mode 1 only)	par_1	AD[31::0] address parity error.	PCI Spec Appendix C, Rule 32 b
34	BX_OBSRULE_PAR_2 (PCI-X Mode 1 only)	par_2	AD[63::32] address parity error.	PCI Spec Appendix C, Rule 32 c
35	BX_OBSRULE_PAR_3 (PCI-X Mode 1 only)	par_3	AD[31::0] data parity error occurred but was not signaled.	PCI-X Spec 5.3.
36	BX_OBSRULE_PAR_4 (PCI-X Mode 1 only)	par_4	AD[63::32] data parity error occurred but was not signaled.	PCI-X Spec 5.3.
37	BX_OBSRULE_PAR_5 (PCI-X Mode 1 only)	par_5	AD[31::0] data parity error occurred.	PCI Spec Appendix C, Rule 32 b
38	BX_OBSRULE_PAR_6 (PCI-X Mode 1 only)	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 (PCI-X Mode 1 only)	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 (PCI-X Mode 1 only)	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)

Bit Pos.	rule	CLI Abbr.	Description	Reference
45	BX_OBSRULE_SEM_6 (PCI-X Mode 1 only)	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)
46	BX_OBSRULE_SEM_7 (PCI-X Mode 1 only)	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
<b>The following rules are observed by E2930 and E2923 testcards only.</b>				
53	BX_OBSRULE_ECC_0	ecc_0	ECC address error detected.	PCI-X Spec 1.12.6 Rule 2
54	BX_OBSRULE_ECC_1	ecc_1	Correctable ECC error detected in data phase.	PCI-X Spec 1.12.6 Rule 2
55	BX_OBSRULE_ECC_2	ecc_2	Uncorrectable ECC error detected in data phase.	PCI-X Spec 1.12.6 Rule 3
56	BX_OBSRULE_ECC_3	ecc_3	Uncorrectable and not signaled ECC error detected in data phase.	PCI-X Spec 1.12.6 Rule 2

## bx\_patttype

prop	CLI Abbreviation	pattern (Signals to be used and their logical combination)	Meaning
BX_PATT_BUS0, BX_PATT_BUS1 BX_PATT_BUS2 BX_PATT_BUS3	bus0 bus1 bus2 bus3	All bus signals and the external trigger signals combined with a logical AND.  BX_PATT_BUS2 and BX_PATT_BUS3 are only available for E2930 and E2923 testcards.	Bus pattern terms.
BX_PATT_OBS0, BX_PATT_OBS1, BX_PATT_OBS2, BX_PATT_OBS3 BX_PATT_OBS4 BX_PATT_OBS5	obs0 obs1 obs2 obs3 obs4 obs5	All observer signals (bus state, decoding state and so forth) combined with a logical AND.  BX_PATT_OBS4 and BX_PATT_OBS5 are only available for E2930 and E2923 testcards.	The observer pattern terms.
BX_PATT_ERR0	err0	The error signals, protocol error, data compare error and split transaction error combined with a logical OR.	Error pattern term.
BX_PATT_CONDO, BX_PATT_COND1	cond0 cond1	Pattern string with reference to bus signals and the external trigger input. See BX_RIBLK_CONDSTART.	Conditional pattern terms.

## bx\_perfcondtype

prop	CLI Abbreviation	Default Value	Meaning
BX_PERFCOND_X	x	"0" (false)	Transition condition for the performance sequencer to move from one state to the next.
BX_PERFCOND_CTRAINC	ctrainc	"0" (false)	Condition to increment counter A (nominator counter).
BX_PERFCOND_CTRBINC	ctrbinc	"0" (false)	Condition to increment counter B (denominator counter).
BX_PERFCOND_FBADDEC	fbadec	"0" (false)	Condition to decrement the feedback counter A.
BX_PERFCOND_FBALOAD	fbaload	"0" (false)	Condition to preload the feedback counter A. This condition has priority over incrementing the feedback counter A.

## bx\_perfgentype

prop (CLI Abbreviation)	val	Value Description
BX_PERFGEN_FBA (fba)	32 bit default: 0xffffffff	Feedback counter A for the performance sequencer.
BX_PERFGEN_CTRAMODE (ctrmode)	default: BX_PERFGEN_CTRAMODE_INC	Counter A is incremented by 1 after each count enable.
	BX_PERFGEN_CTRAMODE_INCBYTEN	If the count enable is active, counter A is incremented by the number of transferred bytes. This can be used in counting the real transferred bytes and not only the data phases.

## bx\_perftrantype

prop	CLI Abbreviation	val	Description
BX_PERFTRAN_STATE	state	0 ... 63 default: 0	State identifier. The sequencer starts at state 0.
BX_PERFTRAN_NEXTSTATE	nextstate	0 ... 63 default: 0	Identifier for the state after the transition condition has been met.

# bx\_porttype

port	portnum	Description
BX_PORT_FASTHIF	0 ... n	<p>Specifies a connection via fast host interface card.</p> <p>The first fast host interface card to be found is assigned to "0", the second to "1", and so forth. The order in which the cards are found is not predictable. After the BestXOpen call, use BestXPing and watch the LEDs to see which card is connected to the session.</p>
BX_PORT_OFFLINE	<p>Result of: <i>assumed capabilities</i> OR <i>assumed hardware</i></p> <p>Values for <i>assumed hardware</i> are: BX_HW_E2929A, BX_HW_E2929A_DEEP, BX_HW_E2922A BX_HW_E2929B, BX_HW_E2929B_DEEP, BX_HW_E2922B, BX_HW_E2930A, BX_HW_E2923A</p>	<p>Specifies the Offline/Demo Mode.</p> <p>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.</p> <p>The port number must be the result of the logical OR-combination of the assumed hardware and the assumed capability codes.</p> <p><b>Example:</b> (BX_HW_E2929A   BX_CAPABILITY_64_BIT)</p>
BX_PORT_RS232	BX_PORT_COM1, BX_PORT_COM2, BX_PORT_COM3, BX_PORT_COM4	<p>Specifies a serial port. The initial baud rate is always 9600 Baud. The port number (portnum) specifies the COM port that is used.</p> <p>This port is not available with E2930 testcards.</p>
BX_PORT_PCI_CONF	32 Bit	<p>Device number of the testcard, as used by PCI BIOS and host bridge.</p> <p>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 22).</p>
BX_PORT_USB	<p>0 ... 127 (for E2930) 128 ... 255 (for E2929)</p>	<p>Specifies a connection via the USB port.</p> <p>The first USB connection to an E2930 is assigned to "0", the second to "1", and so forth.</p> <p>The first USB connection to an E2929 is assigned to "128", the second to "129", and so forth.</p> <p>The order in which the cards are found is not predictable. After the BestXOpen call, use BestXPing and watch the LEDs to see which card is connected to the session.</p>

## 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.
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.
	Controls whether and how often the requester-initiator disconnects its current sequence.	
BX_RIBEH_DISCONNECT (disconnect)	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 <b>not</b> 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 <i>all</i> 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 ... 0x3ffff default: 0  <b>Note:</b> Range is 0x00000 ... 0xfffff for E2922 and E2929 testcards.	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. <b>Note:</b> 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 <b>not</b> use this command. It will automatically be replaced by Memory Read DWORD.
	BX_RIBLK_BUSCMD_DEVID_MSG	Device ID Message.
	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 <b>not</b> use this command. It will automatically be replaced by memreadblock.
	BX_RIBLK_BUSCMD_RESERVED_9 (reserved9)	For programming a PCI-X-compliant behavior, do <b>not</b> 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 <b>not</b> 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. <b>Note:</b> 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 <b>not</b> 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	<p>Values for the seven skip-registers.</p> <p><b>Note:</b> Values MUST be QWORD aligned.</p> <p>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.</p> <p>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 <i>not</i> incremented. Only the data generator (dependent on the external bus address) follows the skipped address.</p>

# 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. <b>Note:</b> 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 <b>not</b> use this option.
	BX_RTBEH_SUBSEQ_TABORT	Available only for E2930 and E2923 testcards.  Signals a target abort.
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.

Property Name	Pattern and Trace Memory Syntax	Width (bit)	Signal Type	Trace Memory (bit pos.)	Pattern Type		
					Bus Pattern	Observer Pattern	Error Pattern
BX_SIG_AD64_2 <sup>1</sup>	AD64_2	32	10X vector	159::128	Yes	-	-
BX_SIG_AD32_2 <sup>1</sup>	AD32_2	32	10X vector	191::160	Yes	-	-
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	-	-



Property Name	Pattern and Trace Memory Syntax	Width (bit)	Signal Type	Trace Memory (bit pos.)	Pattern Type		
					Bus Pattern	Observer Pattern	Error Pattern
BX_SIG_bstate <sup>2</sup>	bstate	4	list	115::112	-	yes	-
BX_SIG_decode <sup>2</sup>	decode	3	list	118::116	-	yes	-
BX_SIG_term <sup>2</sup>	term	3	list	121::119	-	yes	-
BX_SIG_xact_cmd <sup>2</sup>	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	-

<sup>1</sup> Only available for E2930 and E2923 testcards.

<sup>2</sup> For more information, see “Values of List Signals” on page 250.

## 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)

signal	Description	
	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 <b>PCI-X</b> 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#.  <b>Note:</b> 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 <b>PCI</b> 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.

prop (CLI Abbreviation)	Property Description		Clearing the Status
	val	Value Description	
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_CMP_DATAPHASE	32-bit value	Returns the data phase in which the compare error occurred.	
BX_STAT_CMP_DATASUBPHASE	2-bit value	Returns the data subphase in which the compare error occurred.	
BX_STAT_CMP_BEATTR	4-bit value	Returns the byte enables in the corresponding attribute phase	
BX_STAT_CMP_BEDATA	8-bit value	Returns the byte enables in the corresponding data phase	
BX_STAT_CMP_XFERSIZE	1-bit value	bit = 0: 32-bit transfer bit = 1: 64-bit transfer	

prop (CLI Abbreviation)	Property Description		Clearing the Status
	val	Value Description	
BX_STAT_CMP_ICMP	1-bit value	set if the error occurred during an initiator read compare	
BX_STAT_CMP_TCMP	1-bit value	set if the error occurred during a target write compare	
BX_STAT_CMP_RESOURCE	1-bit value	bit = 0: data memory used bit = 1: data generator used	
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
	1	PAR has been inverted.	
BX_STAT_ERR_WRPAR64 (wrpar64)	0 (initial)	PAR64 has not been inverted.	BX_STAT_ERR_WRPAR or BX_STAT_ERR_WRPAR64.
	1	PAR64 has been inverted.	
BX_STAT_IABORT (iabort)	0 (initial)	No initiator abort has occurred.	Cleared automatically with a call to BestXExerciserRun or with BestXStatusClear and
	1	A initiator abort has occurred.	
BX_STAT_TABORT (tabort)	0 (initial)	No target abort has occurred.	BX_STAT_IABORT or BX_STAT_TABORT.
	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. <b>Note:</b> 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.	This bit is cleared by calls to BestXExerciserStop or BestXExerciserRun.
	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.	
	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.	
BX_STAT_OBS_ERR (obserr)	Indicates if the observer has detected a protocol error. Calling BestXStatusClear with this property resets all observer status back to zero (FirstErr and AccuErr).		
	0 (initial)	No protocol error detected so far.	
	1	A protocol error message has been detected. Use the FirstErr and AccuErr properties to determine which protocol errors have occurred.	

prop (CLI Abbreviation)	Property Description		Clearing the Status
	val	Value Description	
BX_STAT_TRC_RUNNING (running)	Indicates if the trace memory is running and capturing data.		These properties are cleared automatically with a BestXTraceRun call.
	0 (initial)	The trace memory analyzer is not running.	
	1	The trace memory analyzer is running. The trace memory switches back to stopped if the trigger counter is down to zero or the user issued the BestXTraceStop command.	
BX_STAT_TRC_TRIGGER (trigger)	Indicates if the trigger sequencer triggered.		
	0 (initial)	No trigger occurred so far.	
	1	The trigger sequencer has triggered.	
BX_STAT_TRC_MEMFULL (memfull)	Indicates if the trace memory was filled at least once completely.		
	0 (initial)	The trace memory has not yet been completely filled.	
	1	The entire trace memory is filled with data.	
BX_STAT_TRC_TRIGPOS (trigpos)	32-bit value	Indicates where the trigger point is in relation to the starting point of the trace memory.	
BX_STAT_TRC_LINES (lines)	32-bit value	The total number of captured lines.	



prop (CLI Abbreviation)	Property Description		Clearing the Status
	val	Value Description	
BX_STAT_RESETCODE (resetcode)	4-bit value	The state of DEVSEL# (bit 0), STOP# (bit 1), TRDY# (bit 2) and PERR# (bit 3) 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. This is the only possible value for E2930 and E2923 testcards.	
	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.	A clear has no effect.
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_PCIXPOWER (pcixpower)	Indicates whether the PCI-X bus power is available. (This property is only useful if the E2929/E2930 is run with external power supply).		
	0	PCI-X bus power is not available.	
	1	PCI-X bus power is available.	
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.	

prop (CLI Abbreviation)	Property Description		Clearing the Status
	val	Value Description	
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.
<p>The following properties providing status information are only available for E2930 testcards.</p> <p><b>Note:</b> If you poll one of these properties repeatedly, make sure to leave at least 0.6 seconds time between subsequent calls. Else the read values will not be updated properly.</p>			
BX_STAT_DIAG_BOARDTEMP		Board temperature (degree celsius)	
BX_STAT_DIAG_V5		Board 5V rail	
BX_STAT_DIAG_V12		PCI 12V rail	
BX_STAT_DIAG_VCORE		ASIC core voltage	
BX_STAT_DIAG_VCC		Board 3.3V rail	
BX_STAT_DIAG_VIO		PCI V(I/O) voltage	
BX_STAT_DIAG_VCCP2		PCI –12V rail	
BX_STAT_DIAG_VBOARD		Board 2.5V rail	
BX_STAT_DIAG_FANSPEED		Fanspeed of ASIC fan in RPS (rotations per second)	

## bx\_tracetype

prop (CLI Abbreviation)	Property Description	
	val	Value Description
BX_TRACE_TRIG_HISTORY (trighistory)	<p>0 ... 0x1ffffff in steps of 4. default: 0x1000000</p> <p>The default depends on the available trace memory space: It is set so that the trigger can be found in the middle of the trace memory.</p>	Trigger counter preload value that defines how many lines are captured after a trigger event.

## bx\_trigcondtype

prop	CLI Abbreviation	Default Value	Meaning
BX_TRIGCOND_X	x	"0" (false)	Transition condition for the trigger sequencer to move from one state to the next.
BX_TRIGCOND_TRIG	trig	"0" (false)	Trigger condition.
BX_TRIGCOND_SQ	sq	"0" (false)	Condition to store the current trace data line in the trace memory (Storage Qualifier Condition).
BX_TRIGCOND_FBADEC	fbadec	"0" (false)	Condition to decrement feedback counter A.
BX_TRIGCOND_FBALOAD	fbaload	"0" (false)	Condition to preload feedback counter A. This condition has priority over incrementing feedback counter A.
BX_TRIGCOND_FBBDEC	fbbddec	"0" (false)	Condition to decrement feedback counter B.
BX_TRIGCOND_FBBLOAD	fbblast	"0" (false)	Condition to preload feedback counter B. This condition has priority over incrementing feedback counter B.

## bx\_triggentype

prop	CLI Abbr.	val	Description
BX_TRIGGEN_FBA	fba	32 bit default: 0xffffffff	Feedback counter A for the trigger sequencer.
BX_TRIGGEN_FBB	fbf	32 bit default: 0xffffffff	Feedback counter B for the trigger sequencer.

## bx\_trigtrantype

prop	CLI Abbreviation	val	Description
BX_TRIGTRAN_STATE	state	0 ... 63 default: 0	State identifier. The sequencer starts at state 0.
BX_TRIGTRAN_NEXTSTATE	nextstate	0 ... 63 default: 0	Identifier for the state after the transition condition has been met.

## 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_FAUST	faust	Faust version (E2930 and E2923 only).
BX_VERSION_MEPHISTO	mephisto	Mephisto version (E2929 and E2922 only).
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 deepttrace board (E2929 and E2922 only).
BX_VERSION_PRODUCT	product	Testcard type: E2930A, E2929B, E2929B_DEEP, E2929A, E2929A_DEEP, E2923A, E2922B or E2922A.
BX_VERSION_XILINX	xilinx	Version number of the Xilinx on the Mainboard.

# 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 “ <i>bxppr_behresulttype</i> ” on page 262).	
	1 ... MaxDWord default: 1	
BXPPR_BEHPERM_TUPLES (tuples)	Maximum number of groups that are permuted 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).
<b>The following properties are only valid for requester-initiator behavior permutation:</b>		
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 $\mu$ s 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 $\mu$ s required to transfer enough data to achieve complete coverage of all required group tuples.
<b>The following properties are only valid for requester-initiator behavior permutation and only if requester-initiator block generation is enabled:</b>		
BXPPR_BEHRES_RUNS (runs)	0 ... $2^{32}$	This property is valid only for requester-initiator behaviors.  If block variations are set to be permuted, 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 permuted, 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 permute blocks and behaviors.	
	default: BXPPR_GEN_ALGORITHM_PERM (pprgenalperm)	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 <b>every</b> 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.	
	<b>Mephisto 3.0:</b> bx_yes ... bx_yes default: bx_yes  <b>Mephisto &gt; 3.0:</b> 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\_ribkgaptype

gapprop (CLI Abbreviation)	Property Description	
	value/pValue (CLI Abbreviation)	Value Description
BXPPR_RIBLK_GAP_BUSADDR_LO (busaddrlo)	Address of the gap.	
	0 ... MaxDWord	
BXPPR_RIBLK_GAP_BUSADDR_HI (busaddrhi)	High (upper 64 bit) part of gap address.	
	0 ... MaxDWord	
BXPPR_RIBLK_GAP_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 (memreadddword)	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 <b>not</b> 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 <b>not</b> 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_RIBLK_GAP_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_ribktype).	
	0000/b ... 1111/b	Refer to BX_RIBLK_BYTEN in <i>"bx_ribktype"</i> on page 241.
	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_RIBLK_GAP_NUMBYTES (numbytes)	Number of bytes that are needed to fill the gap for the current block.	
	0 ... MaxDWord	

# bxppr\_riblkpermtyp

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	

<b>blockprop</b> (CLI Abbreviation)	<b>Property Description</b>	
	<b>value/pValue</b> (CLI Abbreviation)	<b>Value Description</b>
BXPPR_RIBLKPERM_STARTOFFSET (startoffset)	Start offset (see BestXRIBkSet) 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 permuted against each other for a calculation of coverage.	

## bxppr\_riblkresulttype

<b>blkprop</b> (CLI Abbreviation)	<b>value/pValue</b>	<b>Description</b>
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 “ <i>bx_riblktype</i> ” on page 241. 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 <b>not</b> use this command. It will automatically be replaced by memreadblock.
	BX_RIBLK_BUSCMD_ALIAS_MEM_WRITEBLOCK	For programming a PCI-X-compliant behavior, do <b>not</b> 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 “ <i>bx_riblktype</i> ” on page 241.
	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.

blkprop (CLI Abbreviation)	Property Description	
	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

## A

addr\_phase 249  
address space type 209  
administration functions 26  
analysis functions 43  
analyzer functions 115

## B

baud rate 24  
behavior  
    permutation type 261  
    result type 262  
board type 210  
bus pattern term syntax diagram 118

## C

card status functions 32  
completer-initiator  
    behavior type 213  
    generic type 215  
    PPR functions 194  
    programming functions 87  
completer-target  
    behavior type 215  
    generic type 217  
    PPR functions 187  
    programming functions 80  
    split type 218  
condition  
    for preloading the feedback  
        counters 122  
        reference 123  
configuration space programming  
functions 94  
connection to the PCI-X testcard 23  
counter 142  
    identifier 143

## D

data memory functions 108  
decoder  
    programming functions 98  
    property type 220  
    type 221  
denominator counter 143  
device

ID 21  
    identifier format 22  
diagrams  
    bus pattern term syntax 118  
    error pattern term syntax 120  
    for numbers and 10Xnumbers 117  
    observer pattern term syntax 119  
display functions 151  
Documentation Overview 11

## E

error  
    functions 205  
    pattern term syntax diagram 120  
    type 230  
error code  
    BX\_E\_ALIGN 230  
    BX\_E\_BAD\_DECODER\_NUMBER 230  
    BX\_E\_BAD\_FILE\_FORMAT 230  
    BX\_E\_BAD\_HANDLE 230  
    BX\_E\_BAUDRATE 230  
    BX\_E\_CANNOT\_CONNECT 230  
    BX\_E\_CANNOT\_CONNECT\_CORE 230  
exerciser  
    generic type 222  
    memory 15  
expansion ROM programming  
functions 106

## F

fast host interface 237  
functions  
    BestXAllDefaultSet 35  
    BestXAllResourceUnlock 26  
    BestXBoardDefaultSet 35  
    BestXBoardGet 36  
    BestXBoardProg 37  
    BestXBoardRead 38  
    BestXBoardReset 38  
    BestXBoardSet 39  
    BestXCapabilityRead 27  
    BestXCIBehDefaultSet 88  
    BestXCIBehGet 89  
    BestXCIBehMemInit 90  
    BestXCIBehSet 91  
    BestXCIGenDefaultSet 92  
    BestXCIGenGet 92  
    BestXCIGenSet 93  
    BestXCIOpen 20  
    BestXConfRegGet 94

BestXConfRegMaskGet 95  
BestXConfRegMaskSet 96  
BestXConfRegSet 97  
BestXCTBehDefaultSet 81  
BestXCTBehGet 82  
BestXCTBehMemInit 83  
BestXCTBehSet 84  
BestXCTGenDefaultSet 85  
BestXCTGenGet 85  
BestXCTGenSet 86  
BestXCTSplitCondDefaultSet 103  
BestXCTSplitCondGet 104  
BestXCTSplitCondSet 105  
BestXDataMemInit 108  
BestXDataMemRead 109  
BestXDataMemWrite 110  
BestXDevIdentifierGet 21  
BestXDisplayStringWrite 152  
BestXDisplayWrite 153  
BestXErrorStringGet 206  
BestXExerciserBreak 57  
BestXExerciserContinue 57  
BestXExerciserDefaultSet 58  
BestXExerciserGenDefaultSet 58  
BestXExerciserGenGet 59  
BestXExerciserGenSet 60  
BestXExerciserPause 60  
BestXExerciserProg 61  
BestXExerciserRead 61  
BestXExerciserReset 62  
BestXExerciserRun 63  
BestXExerciserStop 63  
BestXExpRomRead 106  
BestXExpRomWrite 107  
BestXHostPCIRegRead 112  
BestXHostPCIRegWrite 113  
BestXInterruptGenerate 42  
BestXLastErrorGet 207  
BestXLastErrorStringGet 207  
BestXMailboxReceiveRegRead 154  
BestXMailboxSendRegWrite 155  
BestXObsAccuErrorGet 47  
BestXObsBitPosGet 54  
BestXObsFirstErrorGet 48  
BestXObsMaskGet 52  
BestXObsMaskSet 51  
BestXObsProg 46  
BestXObsRead 45  
BestXObsRuleGet 53  
BestXObsRuleStringGet 50  
BestXObsStatusGet 49  
BestXOpen 23

BestXPattProg 116  
 BestXPCICfgMailboxReceiveRegRead 156  
 BestXPCICfgMailboxSendRegWrite 157  
 BestXPerfCondSet 141  
 BestXPerfCtrRead 142  
 BestXPerfDefaultSet 143  
 BestXPerfGenDefaultSet 144  
 BestXPerfGenGet 145  
 BestXPerfGenSet 146  
 BestXPerfProg 147  
 BestXPerfRun 148  
 BestXPerfStop 150  
 BestXPerfTranCondDefaultSet 148  
 BestXPerfTranSet 149  
 BestXPerfUpdate 150  
 BestXPing 24  
 BestXPMERead 40  
 BestXPMWrite 41  
 BestXPPrCIBehListDefaultSet 195  
 BestXPPrCIBehListGet 196  
 BestXPPrCIBehListSet 197  
 BestXPPrCIBehPermDefaultSet 198  
 BestXPPrCIBehPermGet 198  
 BestXPPrCIBehPermSet 199  
 BestXPPrCIBehResultGet 200  
 BestXPPrCIDefaultSet 200  
 BestXPPrCTBehListDefaultSet 188  
 BestXPPrCTBehListGet 189  
 BestXPPrCTBehListSet 190  
 BestXPPrCTBehPermDefaultSet 191  
 BestXPPrCTBehPermGet 191  
 BestXPPrCTBehPermSet 192  
 BestXPPrCTBehResultGet 193  
 BestXPPrCTDefaultSet 193  
 BestXPPrDelete 160  
 BestXPPrGenDefaultSet 162  
 BestXPPrGenGet 163  
 BestXPPrGenSet 164  
 BestXPPrInit 161  
 BestXPPrProg 161  
 BestXPPrReportFile 202  
 BestXPPrReportGet 202  
 BestXPPrReportSet 203  
 BestXPPrReportWrite 201  
 BestXPPrRIBehListDefaultSet 166  
 BestXPPrRIBehListGet 167  
 BestXPPrRIBehListSet 168  
 BestXPPrRIBehPermDefaultSet 169  
 BestXPPrRIBehPermGet 169  
 BestXPPrRIBehPermSet 170  
 BestXPPrRIBehResultGet 171  
 BestXPPrRIBlkGapGet 178  
 BestXPPrRIBlkListDefaultSet 172  
 BestXPPrRIBlkListGet 173  
 BestXPPrRIBlkListSet 174  
 BestXPPrRIBlkPermDefaultSet 175  
 BestXPPrRIBlkPermGet 175  
 BestXPPrRIBlkPermSet 176  
 BestXPPrRIBlkResultGet 177  
 BestXPPrRIDefaultSet 177  
 BestXPPrRTBehListDefaultSet 180

BestXPPrRTBehListGet 181  
 BestXPPrRTBehListSet 182  
 BestXPPrRTBehPermDefaultSet 183  
 BestXPPrRTBehPermGet 184  
 BestXPPrRTBehPermSet 185  
 BestXPPrRTBehResultGet 186  
 BestXPPrRTDefaultSet 186  
 BestXPUProg 39  
 BestXResourceIsLocked 28  
 BestXResourceLock 29  
 BestXResourceUnlock 30  
 BestXRIBehSet 68  
 BestXRIBlockDefaultSet 69  
 BestXRIBlockGet 70  
 BestXRIBlockMemInit 70  
 BestXRIBlockSet 71  
 BestXRIGenDefaultSet 72  
 BestXRIGenGet 72  
 BestXRIGenPropGet 72  
 BestXRIGenSet 73  
 BestXRS232BaudRateSet 24  
 BestXRTBehDefaultSet 75  
 BestXRTBehGet 76  
 BestXRTBehMemInit 76  
 BestXRTBehSet 77  
 BestXRTGenDefaultSet 78  
 BestXRTGenGet 78  
 BestXRTGenSet 79  
 BestXStatusClear 32  
 BestXStatusRead 33  
 BestXTDecoderAllSet 99  
 BestXTDecoderDefaultSet 100  
 BestXTDecoderGet 101  
 BestXTDecoderSet 102  
 BestXTraceBitPosGet 132  
 BestXTraceBytePerLineGet 133  
 BestXTraceDataRead 134  
 BestXTraceDefaultWrite 135  
 BestXTraceDump 136  
 BestXTraceRead 137  
 BestXTraceRun 137  
 BestXTraceStop 138  
 BestXTraceWrite 139  
 BestXTrigCondSet 122  
 BestXTrigDefaultSet 125  
 BestXTrigGenDefaultSet 125  
 BestXTrigGenGet 125  
 BestXTrigGenSet 127  
 BestXTrigProg 128  
 BestXTrigTranCondDefaultSet 129  
 BestXTrigTranSet 130  
 BestXVersionRead 31

## G

generic  
   control and setup functions 34  
   exerciser functions 56  
   functions 19  
   performance properties 146  
   PPR functions 162  
   type 263

## H

host access functions 111

## I

initialization and connection functions 20  
 interrupt generation function 42

## L

list type 264

## M

mailbox functions 154

## N

naming conventions 16  
 nominator counter 143

## O

observer pattern term syntax diagram 119  
 observer programming 43  
 observer rule type 231  
 Overview  
   Documentation 11

## P

pattern term  
   function 115  
   operators 117  
 pattern type 235  
 PCI and PCI-X C-API/PPR 14  
 performance  
   condition type 235  
   counter 142  
   generic properties 146  
   generic types 236  
   sequencer programming functions 140  
   transition type 236  
 port 23  
   type 237  
 power management event functions 40  
 PPR  
   administration functions 160  
   completer-initiator functions 194  
   completer-target functions 187  
   generic functions 162  
   reporting functions 201  
   requester-initiator functions 165  
   requester-target functions 179  
 preload performance counter 144  
 programming interfaces 15  
 properties  
   BX\_ADDRSPACE\_CONFIG 209  
   BX\_ADDRSPACE\_IO 209  
   BX\_ADDRSPACE\_MEM 209

BX\_BOARD\_BUSCLOCK 211  
 BX\_BOARD\_DISPLAY 211  
 BX\_BOARD\_M66EN 210  
 BX\_BOARD\_PCIXCAP 210  
 BX\_BOARD\_RESPECTBIOS 211  
 BX\_BOARD\_TRIGIOx\_MODE 211  
 BX\_BOARD\_TRIGIOx\_OUT 212  
 BX\_CAPABILITY\_ANALYZER 27  
 BX\_CAPABILITY\_CAPI 27  
 BX\_CAPABILITY\_EXERCISER 27  
 BX\_CAPABILITY\_OBSERVER 27  
 BX\_CAPABILITY\_PERFORMANCE 27  
 BX\_CIBEH\_CONDSTART 214  
 BX\_CIBEH\_DELAY 214  
 BX\_CIBEH\_ERRMESSAGE 213  
 BX\_CIBEH\_PARTITION 214  
 BX\_CIBEH\_QUEUE 213  
 BX\_CIBEH\_RELREQ 214  
 BX\_CIBEH\_REPEAT 214  
 BX\_CIBEH\_REQ64 214  
 BX\_CIBEH\_STEPS 214  
 BX\_CIGEN\_MESSAGE\_AD 215  
 BX\_CIGEN\_NUMBEH 215  
 BX\_CTBEH\_ACK64 216  
 BX\_CTBEH\_DECSPEED 215  
 BX\_CTBEH\_INITIAL 216  
 BX\_CTBEH\_LATENCY 216  
 BX\_CTBEH\_REPEAT 217  
 BX\_CTBEH\_SPLITENABLE 217  
 BX\_CTBEH\_SPLITLATENCY 217  
 BX\_CTBEH\_SUBSEQ 216  
 BX\_CTBEH\_SUBSEQPHASE 216  
 BX\_CTGEN\_NUMBEH 217  
 BX\_CTSPLIT\_ADDRMASK\_xx 218  
 BX\_CTSPLIT\_ADDRVAL\_xx 218  
 BX\_CTSPLIT\_CMDS 218  
 BX\_CTSPLIT\_DEC 218  
 BX\_CTSPLIT\_QUEUE 219  
 BX\_DEC\_BARx 221  
 BX\_DEC\_CONFIG 221  
 BX\_DEC\_EXPROM 221  
 BX\_DEC\_REQUESTER 221  
 BX\_DECP\_COMPARE 220  
 BX\_DECP\_LOCATION 220  
 BX\_DECP\_PREFETCH 220  
 BX\_DECP\_RESBASE 220  
 BX\_DECP\_RESOURCE 220  
 BX\_DECP\_RESSIZE 220  
 BX\_DECP\_SIZE 220  
 BX\_EGEN\_ARB 228  
 BX\_EGEN\_ARB\_CI 228  
 BX\_EGEN\_ARB\_RI 228  
 BX\_EGEN\_DATAFIX 229  
 BX\_EGEN\_DATAGEN 229  
 BX\_EGEN\_DATASEED 229  
 BX\_EGEN\_ERR\_ECC 224  
 BX\_EGEN\_ERR\_NUM 222  
 BX\_EGEN\_ERR\_PERR 223  
 BX\_EGEN\_ERR\_PHASE 223  
 BX\_EGEN\_ERR\_SERR 224  
 BX\_EGEN\_ERR\_SOURCE 222  
 BX\_EGEN\_ERR\_WRPAR 223  
 BX\_EGEN\_ERR\_WRPAR64 223  
 BX\_EGEN\_INT\_DELAYx 226  
 BX\_EGEN\_INT\_NUM 225  
 BX\_EGEN\_INT\_RDONE 226  
 BX\_EGEN\_INT\_SOURCE 225  
 BX\_EGEN\_PARTCOMP 229  
 BX\_EGEN\_TRIG\_NUM 227  
 BX\_EGEN\_TRIG\_SOURCE 227  
 BX\_INTx 42  
 BX\_OBSRULE\_DEVSEL\_x 232  
 BX\_OBSRULE\_ECC\_x 234  
 BX\_OBSRULE\_FRAME\_x 231  
 BX\_OBSRULE\_INITIATOR\_x 231  
 BX\_OBSRULE\_IRDY\_x 231  
 BX\_OBSRULE\_LAT\_x 232  
 BX\_OBSRULE\_PAR\_0 ... 6 233  
 BX\_OBSRULE\_SEM\_0 ... 13 233  
 BX\_OBSRULE\_STOP\_0 232  
 BX\_OBSRULE\_TARGET\_x 231  
 BX\_OBSRULE\_TRDY\_x 232  
 BX\_OBSRULE\_W64\_x 232  
 BX\_PATT\_BUSx 235  
 BX\_PATT\_CONDX 235  
 BX\_PATT\_ERR0 235  
 BX\_PATT\_OBSx 235  
 BX\_PERFCOND\_CTRINC 235  
 BX\_PERFCOND\_CTRBINC 235  
 BX\_PERFCOND\_FBADEC 235  
 BX\_PERFCOND\_FBALOAD 235  
 BX\_PERFCOND\_X 235  
 BX\_PERFCTR\_A 143  
 BX\_PERFCTR\_B 143  
 BX\_PERFCTR\_REFERENCE 143  
 BX\_PERFGEN\_CTRAMODE 236  
 BX\_PERFGEN\_FBA 236  
 BX\_PERFMEAS 142  
 BX\_PERFTRAN\_NEXTSTATE 236  
 BX\_PERFTRAN\_STATE 236  
 BX\_PORT\_FASTHIF 237  
 BX\_PORT\_OFFLINE 237  
 BX\_PORT\_PCI\_CONF 237  
 BX\_PORT\_RS232 237  
 BX\_PORT\_USB 237  
 BX\_RESLOCK\_ANALYZER 238  
 BX\_RESLOCK\_EXERCISER 238  
 BX\_RESLOCK\_OBSERVER 238  
 BX\_RESLOCK\_PERFORMANCE 238  
 BX\_RIBEH\_BYTECOUNT 239  
 BX\_RIBEH\_DELAY 239  
 BX\_RIBEH\_DISCONNECT 239  
 BX\_RIBEH\_QUEUE 239  
 BX\_RIBEH\_RELREQ 240  
 BX\_RIBEH\_REPEAT 240  
 BX\_RIBEH\_REQ64 240  
 BX\_RIBEH\_SKIP 240  
 BX\_RIBEH\_STEPS 239  
 BX\_RIBEH\_TAG 239  
 BX\_RIBLK\_BUSADDR 241  
 BX\_RIBLK\_BUSADDRHI 241  
 BX\_RIBLK\_BUSCMD 242  
 BX\_RIBLK\_BYTEN 241  
 BX\_RIBLK\_COMPLETION 242  
 BX\_RIBLK\_CONDSTART 241  
 BX\_RIBLK\_DATACMP 243  
 BX\_RIBLK\_INTADDR 241  
 BX\_RIBLK\_NOSNOOP 242  
 BX\_RIBLK\_NUMBYTES 241  
 BX\_RIBLK\_QUEUE 243  
 BX\_RIBLK\_RELAXORDER 242  
 BX\_RIBLK\_RESERVED31 242  
 BX\_RIBLK\_RESOURCE 243  
 BX\_RIGEN\_IABORT 244  
 BX\_RIGEN\_NUMBEH 244  
 BX\_RIGEN\_NUMBLK 244  
 BX\_RIGEN\_REPEATBLK 244  
 BX\_RIGEN\_SKIPREGx 244  
 BX\_RIGEN\_TABORT 244  
 BX\_RTBEH\_ACK64 245  
 BX\_RTBEH\_DECSPEED 245  
 BX\_RTBEH\_INITIAL 245  
 BX\_RTBEH\_LATENCY 245  
 BX\_RTBEH\_REPEAT 246  
 BX\_RTBEH\_SUBSEQ 246  
 BX\_RTBEH\_SUBSEQPHASE 246  
 BX\_RTGEN\_NUMBEH 246  
 BX\_SIG\_ACK64 248  
 BX\_SIG\_ad\_act 249  
 BX\_SIG\_AD32 248  
 BX\_SIG\_AD32\_2 248  
 BX\_SIG\_AD64 248  
 BX\_SIG\_AD64\_2 248  
 BX\_SIG\_addr\_phase\_occ 249  
 BX\_SIG\_bstate 249  
 BX\_SIG\_c\_rule0 249  
 BX\_SIG\_c\_rule1 249  
 BX\_SIG\_CBE3\_0 248  
 BX\_SIG\_CBE7\_4 248  
 BX\_SIG\_ci\_act 249  
 BX\_SIG\_ct\_act 249  
 BX\_SIG\_dcomperr 249  
 BX\_SIG\_decode 249  
 BX\_SIG\_DEVSEL 248  
 BX\_SIG\_FRAME 248  
 BX\_SIG\_gap 249  
 BX\_SIG\_gap\_count 249  
 BX\_SIG\_GNT 248  
 BX\_SIG\_IDSEL 248  
 BX\_SIG\_INTA 248  
 BX\_SIG\_INTB 248  
 BX\_SIG\_INTC 248  
 BX\_SIG\_INTD 248  
 BX\_SIG\_IRDY 248  
 BX\_SIG\_LOCK 248  
 BX\_SIG\_PAR 248  
 BX\_SIG\_PAR64 248  
 BX\_SIG\_pcix\_men 249  
 BX\_SIG\_PERR 248  
 BX\_SIG\_proterr 249  
 BX\_SIG\_REQ 248  
 BX\_SIG\_REQ64 248  
 BX\_SIG\_ri\_act 249  
 BX\_SIG\_ri\_done 249  
 BX\_SIG\_ri\_split\_pending 249  
 BX\_SIG\_RST 248

- BX\_SIG\_rt\_act 249
- BX\_SIG\_SERR 248
- BX\_SIG\_spliterr 249
- BX\_SIG\_STOP 248
- BX\_SIG\_term 249
- BX\_SIG\_TRDY 248
- BX\_SIG\_trigiox 248
- BX\_SIG\_xact\_cmd 249
- BX\_SIG\_xact\_tran64 249
- BX\_SIZE\_BYTE 252
- BX\_SIZE\_DWORD 252
- BX\_SIZE\_WORD 252
- BX\_STAT\_BUSSPEED 257
- BX\_STAT\_BUSWIDTH 257
- BX\_STAT\_CLK 257
- BX\_STAT\_CMP\_ACTUAL\_xx 253
- BX\_STAT\_CMP\_ADDR\_xx 253
- BX\_STAT\_CMP\_ATTR\_LO 253
- BX\_STAT\_CMP\_BEATTR 253
- BX\_STAT\_CMP\_BEDATA 253
- BX\_STAT\_CMP\_CMD 253
- BX\_STAT\_CMP\_DATAPHASE 253
- BX\_STAT\_CMP\_DATASUBPHASE 253
- BX\_STAT\_CMP\_EXTCMD 253
- BX\_STAT\_CMP\_FAIL 253
- BX\_STAT\_CMP\_ICMP 254
- BX\_STAT\_CMP\_REF\_xx 253
- BX\_STAT\_CMP\_RESOURCE 254
- BX\_STAT\_CMP\_TCMP 254
- BX\_STAT\_CMP\_XFERSIZE 253
- BX\_STAT\_DIAG\_BOARDTEMP 258
- BX\_STAT\_DIAG\_FANSPEED 258
- BX\_STAT\_DIAG\_V12 258
- BX\_STAT\_DIAG\_V5 258
- BX\_STAT\_DIAG\_VBOARD 258
- BX\_STAT\_DIAG\_VCC 258
- BX\_STAT\_DIAG\_VCCP2 258
- BX\_STAT\_DIAG\_VCORE 258
- BX\_STAT\_DIAG\_VIO 258
- BX\_STAT\_ERR\_PERR 254
- BX\_STAT\_ERR\_SERR 254
- BX\_STAT\_ERR\_WRPAR 254
- BX\_STAT\_ERR\_WRPAR64 254
- BX\_STAT\_IABORT 254
- BX\_STAT\_INT\_x 254
- BX\_STAT\_INVISIBLE 257
- BX\_STAT\_LASTRESET 258
- BX\_STAT\_OBS\_ERR 255
- BX\_STAT\_PCIRESET 257
- BX\_STAT\_PCIXMODE 257
- BX\_STAT\_PCIXPOWER 257
- BX\_STAT\_PIGGYBACKID 257
- BX\_STAT\_RESETCODE 257
- BX\_STAT\_SPLITFAIL 254
- BX\_STAT\_TABORT 254
- BX\_STAT\_TEST 255
- BX\_STAT\_TRC\_LINES 256
- BX\_STAT\_TRC\_MEMFULL 256
- BX\_STAT\_TRC\_RUNNING 256
- BX\_STAT\_TRC\_TRIGGER 256
- BX\_STAT\_TRC\_TRIGPOS 256
- BX\_STAT\_TRIGIO 257
- BX\_TRACE\_TRIG\_HISTORY 258
- BX\_TRIGCOND\_FBADDEC 259
- BX\_TRIGCOND\_FBALOAD 259
- BX\_TRIGCOND\_FBBDEC 259
- BX\_TRIGCOND\_FBBLOAD 259
- BX\_TRIGCOND\_SQ 259
- BX\_TRIGCOND\_TRIG 259
- BX\_TRIGCOND\_X 259
- BX\_TRIGGEN\_FBA 259
- BX\_TRIGGEN\_FBB 259
- BX\_TRIGTRAN\_NEXTSTATE 260
- BX\_TRIGTRAN\_STATE 260
- BX\_VERSION\_ALTERA 260
- BX\_VERSION\_BOARDID 260
- BX\_VERSION\_CAPI 260
- BX\_VERSION\_CORE 260
- BX\_VERSION\_FIRMWARE 260
- BX\_VERSION\_MEPHISTO 260
- BX\_VERSION\_PRODUCT 260
- BX\_VERSION\_SERIALNO 260
- BX\_VERSION\_TEAM 260
- BX\_VERSION\_XILINX 260
- BXPPR\_BEHPERM\_FIRSTPERM 261
- BXPPR\_BEHPERM\_TUPLES 261
- BXPPR\_BEHRES\_DATA 262
- BXPPR\_BEHRES\_LASTPERM 262
- BXPPR\_BEHRES\_RUNS 262
- BXPPR\_BEHRES\_TIME 262
- BXPPR\_BEHRES\_TUPLES\_DATA 262
- BXPPR\_BEHRES\_TUPLES\_LASTPERM 262
- BXPPR\_BEHRES\_TUPLES\_RUNS 262
- BXPPR\_BEHRES\_TUPLES\_TIME 262
- BXPPR\_GEN\_ADBLIMITATION 264
- BXPPR\_GEN\_ALGORITHM 263
- BXPPR\_GEN\_BUSSPEED 263
- BXPPR\_GEN\_BUSWIDTH 264
- BXPPR\_GEN\_PRESET 263
- BXPPR\_GEN\_SEED 264
- BXPPR\_GEN\_USE\_CIBEH 263
- BXPPR\_GEN\_USE\_CTBEH 263
- BXPPR\_GEN\_USE\_RIBEH 263
- BXPPR\_GEN\_USE\_RIBLK 263
- BXPPR\_GEN\_USE\_RTBEH 263
- BXPPR\_GEN\_XFERCLKS 264
- BXPPR\_REPORT\_CAPI 265
- BXPPR\_REPORT\_CONTENTS 265
- BXPPR\_RIBLK\_ALIGN 270
- BXPPR\_RIBLK\_BUSCMD 270
- BXPPR\_RIBLK\_BYTEN 270
- BXPPR\_RIBLK\_NOSNOOP 271
- BXPPR\_RIBLK\_NUMBYTES 271
- BXPPR\_RIBLK\_RELAXORDER 271
- BXPPR\_RIBLK\_GAP\_BUSADDR\_HI 266
- BXPPR\_RIBLK\_GAP\_BUSADDR\_LO 266
- BXPPR\_RIBLK\_GAP\_BUSCMD 266
- BXPPR\_RIBLK\_GAP\_BYTEN 267
- BXPPR\_RIBLK\_GAP\_NUMBYTES 267
- BXPPR\_RIBLKPERM\_BLOCKSIZE 268
- BXPPR\_RIBLKPERM\_BUSADDRESS\_HI 268
- BXPPR\_RIBLKPERM\_BUSADDRESS\_L 268
- O 268
- BXPPR\_RIBLKPERM\_DIRECTION 268
- BXPPR\_RIBLKPERM\_FILLGAPS 268
- BXPPR\_RIBLKPERM\_FIRSTPERM 268
- BXPPR\_RIBLKPERM\_INTADDR 268
- BXPPR\_RIBLKPERM\_RESOURCE 268
- BXPPR\_RIBLKPERM\_SIZELIMIT 269
- BXPPR\_RIBLKPERM\_STARTOFFSET 269
- BXPPR\_RIBLKPERM\_TUPLES 269
- BXPPR\_RIBLKRES\_ACTUALSIZE 269
- BXPPR\_RIBLKRES\_LASTPERM 269
- BXPPR\_RIBLKRES\_NUMGAPS 269
- protocol
  - observer functions 43
  - permutator and randomizer 15
  - permutator and randomizer functions 159
- protocol observer 43
- R**
- reference counter 143
- report type 265
- requester-initiator
  - behavior type 239
  - block gap type 266
  - block permutation type 268
  - block result type 269
  - block type 241, 270
  - generic type 244
  - PPR functions 165
  - programming functions 64
- requester-target
  - behavior type 245
  - generic type 246
  - PPR functions 179
  - programming functions 74
- resource
  - locks 28
  - type 238
- S**
- serial interface 237
- signal type 247
- size type 252
- standard analysis functions 43
- status type 253
- storage qualifier condition 122
- structure of the PCI-X C-API/PPR reference 17
- syntax diagrams for numbers and 10Xnumbers 117
- T**
- terminology changes 14
- trace memory
  - functions 131
  - trigger sequencer functions 121

- trace type 258
- transition condition 122
- trigger
  - condition 122
  - condition type 259
  - generic type 259
  - sequencer programming functions 121
  - transition type 260
- type definitions 209
- types
  - bx\_addrspacetype 209
  - bx\_boardtype 210
  - bx\_cibehtype 213
  - bx\_cigentype 215
  - bx\_ctbehtype 215
  - bx\_ctgentype 217
  - bx\_ctsplittype 218
  - bx\_decproptype 220
  - bx\_dectype 221
  - bx\_egentype 222
  - bx\_errtype 230
  - bx\_obsruletype 231
  - bx\_patttype 235
  - bx\_perfcondtype 235
  - bx\_perfgentype 236
  - bx\_perfrantype 236
  - bx\_porttype 237
  - bx\_resourcetype 238
  - bx\_ribehtype 239
  - bx\_riblkttype 241
  - bx\_rigentype 244
  - bx\_rtbehtype 245
  - bx\_rtgentype 246
  - bx\_signaltype 247
  - bx\_sizetype 252
  - bx\_statustype 253
  - bx\_tracetype 258
  - bx\_trigcondtype 259
  - bx\_triggentype 259
  - bx\_trigtrantype 260
  - bx\_versiontype 260
  - bxppr\_behpermttype 261
  - bxppr\_behresulttype 262
  - bxppr\_gentype 263
  - bxppr\_listtype 264
  - bxppr\_reporttype 265
  - bxppr\_ribkgaptype 266
  - bxppr\_riblkpermttype 268
  - bxppr\_riblkresulttype 269
  - bxppr\_riblkttype 270

## V

---

- vendor ID 21
- version type 260

## X

---

- xact\_attr\_ad 249
- xact\_attr\_cbe 249
- xact\_dac 249
- xact\_lock 249
- xact\_trigio0 249
- xact\_trigio1 249
- xact\_trigio2 249
- xact\_trigio3 249

