



Tcl Development Guide

Version 1.0

Release 6.80 EA

December 2014



Copyright © 2014 Ixia. All rights reserved.

This publication may not be copied, in whole or in part, without Ixia's consent.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19.

Ixia, the Ixia logo, and all Ixia brand names and product names in this document are either trademarks or registered trademarks of Ixia in the United States and/or other countries. All other trademarks belong to their respective owners.

The information herein is furnished for informational use only, is subject to change by Ixia without notice, and should not be construed as a commitment by Ixia. Ixia assumes no responsibility or liability for any errors or inaccuracies contained in this publication.

Corporate Headquarters	Ixia Worldwide Headquarters 26601 W. Agoura Rd. Calabasas, CA 91302 USA +1 877 FOR IXIA (877 367 4942) +1 818 871 1800 (International) (FAX) +1 818 871 1805 sales@ixiacom.com	Web site: www.ixiacom.com General: info@ixiacom.com Investor Relations: ir@ixiacom.com Training: training@ixiacom.com Support: support@ixiacom.com +1 818 595 2599 For the online support form, go to: http://www.ixiacom.com/support/inquiry/
EMEA	Ixia Technologies Europe Limited Part 2nd floor, Clarion House, Norreys Drive Maidenhead, UK SL6 4FL +44 (1628) 408750 FAX +44 (1628) 639916 salesemea@ixiacom.com	Support: support-emea@ixiacom.com +40 21 301 5699 For the online support form, go to: http://www.ixiacom.com/support/inquiry/?location=emea
Asia Pacific	Ixia Pte Ltd 210 Middle Road #08-01 IOI Plaza Singapore 188994	Support: support-asiapac@ixiacom.com +91 80 4939 6410 For the online support form, go to: http://www.ixiacom.com/support/inquiry/
Japan	Ixia Communications KK Nishi-Shinjuku Mitsui Bldg 11F 6-24-1, Nishi-Shinjuku, Shinjuku-ku Tokyo 160-0023 Japan	Support: support-japan@ixiacom.com +81 3 5326 1980 For the online support form, go to: http://www.ixiacom.com/support/inquiry/
India	Ixia Technologies Pvt Ltd Tower 1, 7th Floor, UMIYA Business Bay Cessna Business Park Survey No. 10/1A, 10/2, 11 & 13/2 Outer Ring Road, Varthur Hobli Kadubeesahalli Village Bangalore East Taluk Bangalore-560 037, Karnataka, India +91 80 42862600	Support: support-india@ixiacom.com +91 80 4939 6410 For the online support form, go to: http://www.ixiacom.com/support/inquiry/?location=india

China	Ixia Technologies (Shanghai) Company Ltd Unit 3, 11th Floor, Raffles City, Beijing Beijing, 100007 P.R.C.	Support: support-china@ixiacom.com 400 898 0598 (Greater China Region) +86 10 5732 3932 (Hong Kong)
-------	---	--

Change: 4150211, Date: September 24, 2013

For viewing the FAQs related to the product, go to Ixia Technical Support Online:
https://ebsoprod.ixiacom.com/OA_HTML/jtflogin.jsp

Table of Contents

About This Guide

Purpose	xxxv
Manual Content	xxxv
Related Documentation	xxxvi
Technical Support	xxxvi

Chapter 1 Tcl API Overview

ScriptGen	1-4
What's New in Version 6.80 EA?	1-4
All Deprecated Commands and Options	1-5

Chapter 2 Quick Start

Installing the IxOS Tcl Client	2-1
To install the IxOS TCL Client, do the following:	2-2
UNIX Environment	2-4
Windows Environment	2-5

Installing Platform Independent package - tarball	2-6
IxSampleTcl Test Program	2-7
<hr/>	
Chapter 3 High-Level and Utility API Description	
Initialization, Setup and Cleanup	3-3
Mapping and Port Lists	3-3
map	3-5
ixCreatePortListWildCard	3-5
ixCreateSortedPortList	3-6
getAllPorts, getRxPorts and getTxPorts	3-6
Including Source Code	3-7
ixSource	3-7
Chassis and TclServer Connection	3-7
ixConnectToTclServer / ixDisconnectTclServer	3-7
ixProxyConnect	3-7
ixConnectToChassis / ixDisconnectFromChassis	3-8
ixGetChassisID	3-8
user	3-8
General Purpose Commands	3-9
ixWriteConfigToHardware	3-9
ixWritePortsToHardware	3-9
cleanUp	3-9
cleanUp	3-9
Port Ownership	3-10
ixLogin / ixLogout	3-10
ixCheckOwnership	3-10
ixPortTakeOwnership / ixTakeOwnership / ixPortClearOwnership / ixClearOwnership	3-10
Data Transmission	3-12
Setup	3-13
ixCheckLinkState	3-13
ixCheckPPPState	3-13
ixSetPortPacketFlowMode / ixSetPacketFlowMode	3-13
ixSetPortPacketStreamMode / ixSetPacketStreamMode	3-13
ixSetPortAdvancedStreamSchedulerMode / ixSetAdvancedStreamScheduler- Mode	3-14
ixSetPortTcpRoundTripFlowMode / ixSetTcpRoundTripFlowMode	3-14

disableUdfs	3-14
Negotiation	3-15
ixRestartPortAutoNegotiation / ixRestartAutoNegotiation	3-15
ixRestartPortPPPNegotiation / ixRestartPPPNegotiation	3-15
Start Transmit	3-15
ixStartPortTransmit / ixStartTransmit / ixStopPortTransmit / ixStopTransmit ..	3-15
ixStartStaggeredTransmit	3-16
ixCheckPortTransmitDone / ixCheckTransmitDone	3-16
ixStartPortCollisions / ixStartCollisions / ixStopPortCollisions / ixStopCollisions ..	3-16
ixStartPortAtmOamTransmit / ixStartAtmOamTransmit / ixStopPortAtmOamTransmit / ixStopAtmOamTransmit	3-17
ixClearScheduledTransmitTime / ixSetScheduledTransmitTime	3-17
ixLoadPoePulse / ixLoadPortPoePulse	3-17
Calculation Utilities	3-17
calculateMaxRate	3-17
host2addr	3-18
byte2IpAddr	3-18
dectohex	3-18
hextodec	3-18
Data Capture and Statistics	3-19
 Setup	3-19
ixSetPortCaptureMode / ixSetCaptureMode	3-19
ixSetPortPacketGroupMode / ixSetPacketGroupMode	3-20
ixSetPortDataIntegrityMode / ixSetDataIntegrityMode	3-20
ixSetPortSequenceCheckingMode / ixSetSequenceCheckingMode	3-20
ixClearTimeStamp	3-21
ixClearPortStats / ixClearStats	3-21
ixClearPortPacketGroups/ ixClearPacketGroups	3-21
ixResetSequenceIndex/ ixResetPortSequenceIndex	3-21
 Capture Data	3-22
ixStartPortCapture / ixStartCapture / ixStopPortCapture / ixStopCapture ..	3-22
ixStartPortPacketGroups / ixStartPacketGroups / ixStopPortPacketGroups / ixStopPacketGroups	3-22
 Statistics	3-22
ixCollectStats	3-22
ixRequestStats	3-23
ARP	3-24
ixEnableArpResponse / ixEnablePortArpResponse	3-24
ixDisableArpResponse / ixDisablePortArpResponse	3-24
ixClearPortArpTable / ixClearArpTable	3-24
ixTransmitPortArpRequest / ixTransmitArpRequest	3-25

Console Output and Logging	3-26
Error messages	3-26
ixErrorInfo	3-26
errorMsg	3-26
Console Output	3-26
ixPuts	3-26
Logging	3-26
logOn / logOff	3-26
logMsg	3-27
enableEvents	3-27
Port CPU Control	3-28
Issue Port CPU Commands	3-28
issuePcpuCommand	3-28
Miscellaneous Commands	3-29

Chapter 4 Programming

API Structure and Conventions	4-1
Standard Sub-Commands	4-1
Standard Return Codes	4-3
Sequence of Steps	4-5
How to write efficient scripts	4-11
Multi-Client Usage	4-12
Mpexpr versus Expr	4-12

Chapter 5 IxTclHal API Description

Chassis, Cards and Ports	5-2
session	5-4
version	5-4
chassisChain	5-4
timeServer	5-4

chassis	5-5
card	5-6
port	5-6
MII	5-10
mii	5-11
miae	5-11
mmd	5-12
mmdRegister	5-12
xauI	5-13
Packet over Sonet	5-13
sonet	5-14
sonetError	5-15
sonetOverhead	5-16
dcc	5-16
RPR	5-16
rprRingControl	5-17
rprFairness	5-18
rprProtection	5-18
rprOam	5-19
rprTopology	5-20
rprTlvIndividualBandwidth and rprTlvBandwidthPair	5-20
rprTlvWeight	5-21
rprTlvTotalBandwidth	5-21
rprTlvNeighborAddress	5-22
rprTlvStationName	5-22
rprTlvVendorSpecific	5-22
GFP	5-23
gfp	5-23
gfpOverhead	5-23
ppp and pppStatus	5-23
hdlc	5-25
frameRelay	5-26
bert and bertErrorGeneration	5-27
bertUnframed	5-29
ATM	5-29
atmPort	5-30
atmHeader	5-31
atmHeaderCounter	5-31
atmOam	5-32
atmOamActDeact	5-33
atmOamAis	5-33
atmOamFaultManagementCC	5-34
atmOamFaultManagementLB	5-34
atmOamRdi	5-34

atmOamTrace	5-34
Circuit	5-35
sonetCircuit.....	5-35
sonetCircuitList.....	5-36
sonetCircuitProperties.....	5-37
lcas	5-38
10GE	5-39
Link Fault Signaling	5-39
linkFaultSignaling	5-39
customOrderedSet	5-40
txRxPreamble	5-40
Optical Digital Wrapper / FEC	5-41
opticalDigitalWrapper	5-41
fecError	5-41
CDL Support	5-42
cdlPreamble	5-42
xfp	5-43
lasi	5-43
Power Over Ethernet	5-43
poePoweredDevice	5-44
poeAutoCalibration	5-45
poeSignalAcquisition	5-46
portGroup	5-47
Data Transmission	5-49
Streams and Flows	5-49
stream	5-50
streamRegion	5-54
weightedRandomFramesize	5-55
flexibleTimestamp	5-56
Frame Data	5-56
udf	5-56
tableUdf	5-58
tableUdfColumn	5-59
tcpRoundTripFlows	5-59
packetGroup	5-60
dataIntegrity	5-63
Sequence Checking	5-64
forcedCollisions	5-64
Protocols	5-66
protocol	5-66
protocolOffset	5-66
fibreChannel	5-67
fcoe	5-67

Data Link Layer5-67
isl5-67
vlan5-67
stackedVlan5-68
mpls5-69
mplsLabel5-69
IPX5-69
ARP5-71
IP5-72
ipV65-74
ipV6Authentication5-76
ipV6Destination5-77
ipV6Fragment5-77
ipV6HopByHop5-77
ipV6Routing5-78
tcp5-79
udp5-80
igmp5-80
igmpGroupRecord5-81
icmp5-81
rip5-82
ripRoute5-82
dhcp5-83
ptp5-84
pauseControl5-84
srpArp5-85
srplps5-85
srpDiscovery5-86
srpMacBinding5-86
srpUsage5-87
streamQueueList5-87
streamQueue5-88
nativ5-88
Data Capture and Statistics5-89
filter5-89
filterPalette5-91
capture5-92
captureBuffer5-92
qos5-95
atmReassembly5-95
atmFilter5-96
stat5-96

statGroup, statList and statWatch	5-97
packetGroupStats	5-99
latencyBin	5-101
vsrStat	5-103
vsrError	5-103
atmStat	5-105
streamTransmitStats	5-105
Interface Table	5-108
protocolServer	5-108
Interface Table	5-109
interfaceTable	5-110
interfaceEntry	5-112
interfacelpV4	5-113
interfacelpV6	5-113
discoveredList	5-113
discoveredAddress	5-114
discoveredNeighbor	5-114
Using DHCP with Interfaces	5-115
dhcpV4Properties	5-115
dhcpV4DiscoveredInfo	5-116
dhcpV4Tlv	5-117
Using DCHPv6 with Interfaces	5-117
dhcpV6Properties	5-117
dhcpV6DiscoveredInfo	5-118
dhcpV6Tlv	5-119
Using PTP with Interfaces	5-119
Using Fibre Channel and FCoE	5-119
fibreChannel	5-119
fcoe	5-119
IP	5-119
ipAddressTable	5-119
ipAddressTableItem	5-121
Interface Table versus IP Address Table	5-121
sfpPlus	5-122
Port CPU Control	5-123
Port CPU Control	5-123
Issue Port CPU Command	5-123
pcpuCommandService	5-123
serviceManager	5-124

Control File Format5-125
Data Files5-126
serviceManager.....	.5-126

Appendix A IxTclHAL Commands

arpA-2
associationHeader.....	.A-9
atmFilterA-10
atmHeader.....	.A-12
atmHeaderCounterA-15
atmOam.....	.A-18
atmOamActDeactA-25
atmOamAisA-27
atmOamFaultManagementCCA-28
atmOamFaultManagementLBA-29
atmOamRdiA-31
atmOamTraceA-32
atmPortA-35
atmReassemblyA-37
atmStatA-40
autoDetectInstrumentationA-45

basicLinkServices	A-49
bert	A-51
bertErrorGeneration	A-59
bertUnframed	A-62
capture	A-67
captureBuffer	A-73
card	A-79
resourceGroupEx	A-93
cdlPreamble	A-95
cfpPort	A-97
chassis	A-99
chassisChain	A-107
collisionBackoff	A-109
conditionalStats	A-110
conditionalTable	A-114
customOrderedSet	A-116
dataIntegrity	A-117
dcc	A-122
dhcp	A-125

dhcpV4DiscoveredInfo	A-133
dhcpV4Properties	A-135
dhcpV4Tlv	A-137
dhcpV6DiscoveredInfo	A-138
dhcpV6Properties	A-140
dhcpV6Tlv	A-142
discoveredAddress	A-143
discoveredList	A-144
discoveredNeighbor	A-146
encHeader	A-147
espHeader	A-149
extendedLinkServices	A-150
fcEOF	A-156
fcNameServer	A-158
fcNameServerQuery	A-160
fcoe	A-162
fcoeDiscoveredInfo	A-174
fcoeNameServer	A-176
fcoePlogi	A-178

fcoeProperties	A-179
fcPlogi	A-186
fcPort	A-187
fcProperties	A-190
fcSOF	A-192
fecError	A-194
fibreChannel	A-199
filter	A-218
filterPallette	A-228
fipTlv	A-240
flexibleTimestamp	A-241
forcedCollisions	A-244
frameRelay	A-247
gfp	A-251
gfpOverhead	A-256
gre	A-258
hdlc	A-261
icmp	A-267
icmpV6	A-271

icmpV6Error	A-275
icmpV6Informational	A-277
icmpV6MulticastListener	A-278
icmpV6NeighborDiscovery	A-279
icmpV6OptionLinkLayerDestination	A-282
icmpV6OptionLinkLayerSource	A-283
icmpV6OptionMaxTransmissionUnit	A-284
icmpV6OptionPrefixInformation	A-285
icmpV6OptionRedirectedHeader	A-287
icmpV6OptionUserDefine	A-288
icmpV6UserDefine	A-289
IFRHeader	A-290
igmp	A-292
igmpGroupRecord	A-297
interfaceEntry	A-299
interfaceIpV4	A-303
interfaceIpV6	A-304
interfaceTable	A-305
ip	A-320

ipAddressTable	A-331
ipAddressTableItem	A-333
ipV6	A-335
ipV6Address	A-347
ipV6Authentication	A-352
ipV6Destination	A-354
ipV6Fragment	A-356
ipV6HopByHop	A-358
ipV6OptionPAD1	A-361
ipV6OptionPADN	A-362
ipV6OptionJumbo	A-363
ipV6OptionRouterAlert	A-364
ipV6OptionBindingUpdate	A-365
ipV6OptionBindingAck	A-367
ipV6OptionHomeAddress	A-368
ipV6OptionBindingRequest	A-369
ipV6OptionMIPV6UniqueIdSub	A-370
ipV6OptionMIPV6AlternativeCoaSub	A-371
ipV6OptionUserDefine	A-372

ipV6Routing	A-373
ipx	A-375
isl	A-382
lasi	A-385
latencyBin	A-388
lcas	A-389
linkFaultSignaling	A-391
macSecChannel	A-396
macSecRx	A-398
macSecTag	A-400
macSecTx	A-404
mii	A-406
miiae	A-411
mmd	A-414
mmdRegister	A-415
mpls	A-416
mplsLabel	A-420
networkHeader	A-422
npivProperties	A-427

oamEventNotification	A-430
oamEventOrgTlv	A-432
oamFrameTlv	A-433
oamFramePeriodTlv	A-435
oamHeader	A-437
oamInformation	A-442
oamLocallInformationTlv	A-444
oamLoopbackControl	A-446
oamOrganizationSpecific	A-447
oamOrganizationSpecificTlv	A-448
oamPort	A-449
oamRemoteInformationTlv	A-452
oamStatus	A-454
oamSummaryTlv	A-456
oamSymbolPeriodTlv	A-458
oamVariableRequest	A-460
oamVariableRequestTlv	A-462
oamVariableResponse	A-463
oamVariableResponseTlv	A-465

opticalDigitalWrapper	A-467
packetGroup	A-469
packetGroupStats	A-482
packetGroupThresholdList	A-488
packetLengthInsertion	A-491
pauseControl	A-494
pcsLaneError	A-498
pcsLaneStatistics	A-500
pcpuCommandService	A-504
poeAutoCalibration	A-508
poePoweredDevice	A-510
poeSignalAcquisition	A-518
port	A-521
portCpu	A-555
portGroup	A-557
ppp	A-563
pppStatus	A-570
prbsCapture	A-574
protocol	A-576

protocolOffset.....	A-580
protocolServer.....	A-582
protocolPad	A-585
ptp	A-586
ptpAnnounce	A-592
ptpDelayRequest	A-595
ptpDelayResponse	A-596
ptpDiscoveredInfo	A-597
ptpFollowUp	A-599
ptpProperties	A-600
ptpSync	A-603
qos	A-604
rip	A-608
ripRoute	A-612
rprFairness	A-614
rprOam	A-622
rprProtection	A-625
rprRingControl	A-628
rprTlvBandwidthPair	A-631

rprTlvIndividualBandwidth	A-632
rprTlvNeighborAddress	A-634
rprTlvStationName	A-636
rprTlvTotalBandwidth	A-638
rprTlvVendorSpecific	A-640
rprTlvWeight	A-642
rprTopology	A-644
sequenceNumberUdf	A-647
serviceManager	A-650
session	A-653
sfpPlus	A-655
sonet	A-657
sonetCircuit	A-663
sonetCircuitList	A-665
sonetCircuitProperties	A-672
sonetError	A-674
sonetOverhead	A-679
splitPacketGroup	A-682
srpArp	A-685

srpDiscovery	A-690
srplps	A-693
srpMacBinding	A-696
srpUsage	A-697
stackedVlan	A-700
stat	A-703
statAggregator	A-738
statGroup	A-741
statList	A-743
statWatch	A-744
stream	A-747
streamExtractorFilter	A-768
streamExtractorModifier	A-773
streamQueue	A-775
streamQueueList	A-778
streamRegion	A-780
streamTransmitStats	A-782
tableUdf	A-786
tableUdfColumn	A-792

TCLTransceiver	A-794
tcp	A-802
tcpRoundTripFlow	A-806
timeServer	A-811
txLane	A-817
txRxPreamble	A-819
udf	A-822
udp	A-831
usb	A-836
version	A-838
VFTHeader	A-840
vlan	A-841
vsrError	A-845
vsrStat	A-851
weightedRandomFramesize	A-855
xauI	A-860
xfp	A-862

Appendix B Utility Commands

byte2IpAddr	B-2
calculateFPS	B-3
calculateGapBytes	B-4
calculateMaxRate	B-5
calculatePercentMaxRate	B-6
.cleanUp	B-9
clearAllMyOwnership	B-10
dectohex	B-11
disableUdfs	B-12
enableEvents	B-13
errorMsg	B-14
getErrorString	B-15
getStatusLabel	B-16
hextodec	B-17
host2addr	B-18
logMsg	B-19
logOff	B-20

logOn	B-21
mpexpr	B-22
showCmd	B-23
user	B-24

Appendix C High-Level API

getAllPorts	C-2
getRxPorts	C-3
getTxPorts	C-4
issuePcpuCommand	C-5
ixAbortPoeArm	C-7
ixAbortPortPoeArm	C-8
ixArmPoeTrigger	C-9
ixArmPortPoeTrigger	C-10
ixCheckLinkState	C-11
ixCheckOwnership	C-13
ixCheckPPPState	C-15
SixCheckPortTransmitDone	C-17
ixCheckTransmitDone	C-19

ixClearArpTable	C-22
ixClearOwnership	C-24
ixClearPacketGroups	C-26
ixClearPerStreamTxStats	C-29
ixClearPortArpTable	C-30
ixClearPortPacketGroups	C-32
ixClearPortStats	C-34
ixClearScheduledTransmitTime	C-36
ixClearStats	C-37
ixClearTimeStamp	C-39
ixCollectStats	C-41
ixConnectToChassis	C-44
ixConvertFromSeconds	C-46
ixConnectToTclServer	C-47
ixConvertToSeconds	C-48
ixCreatePortListWildCard	C-49
ixCreateSortedPortList	C-51
ixDisableArpResponse	C-52
ixDisablePortArpResponse	C-54

ixDisconnectFromChassis	C-56
ixDisconnectTclServer	C-57
ixEnableArpResponse	C-58
ixEnablePortArpResponse	C-60
ixEnableIntrinsicLatencyAdjustment	C-62
ixEnablePortIntrinsicLatencyAdjustment	C-63
ixErrorInfo	C-64
ixGetChassisID	C-65
ixGetLineUtilization	C-66
ixGlobalSetDefault	C-67
ixInitialize	C-68
ixIsIntrinsicLatencyAdjustmentEnabled	C-70
ixIsOverlappingIpAddress	C-71
ixIsSameSubnet	C-72
ixIsValidHost	C-73
ixIsValidNetMask	C-74
ixIsValidUnicastIp	C-75
ixLoadPoePulse	C-76
ixLoadPortPoePulse	C-77

ixLogin	C-78
ixLogout	C-79
ixMiiConfig utilities	C-80
ixPortClearOwnership	C-82
ixPortTakeOwnership	C-84
ixProxyConnect	C-86
ixPuts	C-88
ixRequestStats	C-89
ixResetPortSequenceIndex	C-90
ixResetSequenceIndex	C-92
ixRestartAutoNegotiation	C-94
ixRestartPortAutoNegotiation	C-95
ixRestartPortPPPAutoNegotiation	C-96
ixRestartPPPNegotiation	C-97
ixSetAdvancedStreamSchedulerMode	C-98
ixSetAutoDetectInstrumentationMode	C-100
ixSetCaptureMode	C-101
ixSetDataIntegrityMode	C-103
ixSetPacketFlowMode	C-105

ixSetPacketGroupMode	C-107
ixSetPacketStreamMode	C-109
ixSetPortAdvancedStreamSchedulerMode	C-111
ixSetPortCaptureMode	C-113
ixSetPortDataIntegrityMode	C-115
ixSetPortPacketFlowMode	C-117
ixSetPortPacketGroupMode	C-119
ixSetPortPacketStreamMode	C-121
ixSetPortSequenceCheckingMode	C-123
ixSetPortTcpRoundTripFlowMode	C-125
:: ixSetScheduledTransmitTime	C-127
ixSetSequenceCheckingMode	C-128
ixSetTcpRoundTripFlowMode	C-130
For example, :: ixSimulatePhysicalInterfaceDown	C-132
ixSimulatePhysicalInterfaceUp	C-133
ixSimulatePortPhysicalInterfaceDown	C-134
ixSimulatePortPhysicalInterfaceUp	C-135
ixSource	C-136
ixStartAtmOamTransmit	C-137

ixStartCapture	C-138
ixStartCollisions	C-140
ixStartPacketGroups	C-143
ixStartPortAtmOamTransmit	C-146
ixStartPortCapture	C-147
ixStartPortCollisions	C-149
ixStartPortPacketGroups	C-151
ixStartPortTransmit	C-154
ixStartStaggeredTransmit	C-156
ixStartTransmit	C-158
ixStopAtmOamTransmit	C-160
ixStopCapture	C-161
ixStopCollisions	C-163
ixStopPacketGroups	C-166
ixStopPortAtmOamTransmit	C-169
ixStopPortCapture	C-170
ixStopPortCollisions	C-172
ixStopPortPacketGroups	C-174
ixStopPortTransmit	C-177

ixStopTransmit	C-179
ixTakeOwnership	C-181
ixTransmitArpRequest	C-183
ixTransmitPortArpRequest	C-185
ixUtils	C-187
ixWriteConfigToHardware	C-188
ixWritePortsToHardware	C-190
map	C-191

Appendix D IxTcl Server Usage

IxTcl Server	D-1
Installation and Invocation	D-1
IxTcl Server Usage	D-2
Options	D-3
Advanced Usage	D-5

Appendix E Reserved Keywords

About this Guide

The information in this section is provided to help you navigate this guide and make better use of its content. A list of related documentation is also included.

Purpose

This guide describes the structure and conventions of the IxExplorer Tcl API and provides detailed information on all API commands. Information is provided on protocol support and indicates the commands, sub-commands, options, and statistics that apply to each protocol.

Manual Content

This guide contains the following sections:

Section	Description
About this Guide	Provides information on this guide, including its purpose, content, and related documentation. Also explains how to contact technical support.
Chapter 1, <i>Tcl API Overview</i>	Provides a brief overview of the Tcl API and the features that are new to this release.
Chapter 2, <i>Quick Start</i>	An overview of a complete, useful Tcl example program. Using this, the basic flow of programming and operation can be viewed.
Chapter 3, <i>High-Level and Utility API Description</i>	Organizes the High-Level and Utility APIs into related discussion groups and describes how to use them at a high level.
Chapter 4, <i>Programming</i>	Explains the basic structure and operation of all of the Tcl Commands.
Chapter 5, <i>IxTclHal API Description</i>	Organizes the APIs into related discussion groups and describes how to use them at a high level.
Appendix A, <i>IxTclHAL Commands</i>	An alphabetical set of reference sheets for all non-protocol related Tcl Commands.
Appendix B, <i>Utility Commands</i>	An alphabetical set of reference sheets for additional test related commands.

Section	Description
Appendix C, <i>High-Level API</i>	Commands which perform a combination of functions against a number of ports.
Appendix D, <i>IxTcl Server Usage</i>	Explains the usage of Tcl Server.
Appendix E, <i>Reserved Keywords</i>	Provides the keywords that are used in IxOS setup. These keywords should not be used as variable names in customer scripts, failing which they will conflict with code execution and exhibit unwanted behavior.

Related Documentation The following guides may help you learn more about Tcl API for IxExplorer. The guides are available on the CD shipped with the application, as well as on the Ixia website at www.ixiacom.com.

- *IxExplorer User Guide*. Details the usage of the IxExplorer GUI for operation with an Ixia chassis and Ixia load modules.
- *Ixia Platform Reference Guide*. Provides a detailed list of all currently supported Ixia chassis and Ixia load modules, as well as general information regarding various technologies covered by Ixia products.
- *IxServer User Guide*. Details the usage of the IxServer GUI for operation on an Ixia chassis.

Technical Support You can obtain technical support for any Ixia product by contacting Ixia Technical Support by any of the methods mentioned on the inside cover of this guide. Technical support from Ixia's corporate headquarters is available Monday through Friday from 06:00 to 18:00, Pacific Standard Time (excluding American holidays). Technical support from Ixia's EMEA and India locations is available Monday through Friday, 08:00 to 17:00 local time (excluding local holidays).

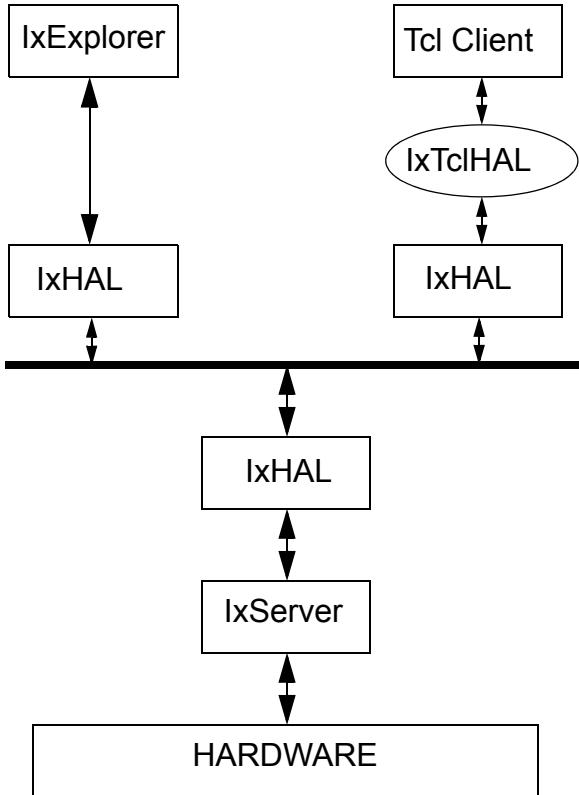
1

Tcl API Overview

The Ixia Tcl Command library provides full access to the Ixia hardware platform. Configurations can be sent to the hardware and various programs can be created and executed on the system. Tcl scripting allows automation of testing procedures when tens to thousands of ports are involved. Ixia's Tcl Command Library is built using a combination of commands that are written in Tcl and commands that are implemented in C/C++. The figure below shows the location of the C++ API Client (IxTclHAL) in the overall picture of the Ixia hardware platform.

NOTE: TCL version 8.5 is supported instead of version 8.4.

Figure 1-1. System Overview Diagram



The IxServer module resides on the computer connected to the test hardware and is responsible for control and operation of the hardware. A single IxServer module exists per chassis.

The IxHAL (Hardware Abstraction Layer) is a C++ based application that provides a higher level abstraction of the Ixia hardware. Working with IxServer, it operates the hardware chassis, cards and ports. When the test software (IxExplorer, IxAutomate (formerly IxScriptMate), Tcl based applications) reside on a different computer than the test hardware, an additional IxHAL copy resides on the remote machine. These two copies act in concert to provide a single interface to upper layers of software.

IxHAL serves as a buffer for configuration information, saving and buffering this data until it receives a command to transfer the data to or from the hardware through IxServer. The IxExplorer software, for example, uses its copy of IxHAL to hold configuration data until it is transferred to the hardware.

In the case of Tcl applications, the Tcl Command Library is a set of Tcl commands that are used to configure the traffic generation, capture and statistics parameters on the Ixia hardware platform. Tcl applications use these commands to configure test parameters and then use a ‘set’ option to transfer the information

into IxHAL. A ‘write’ option causes IxHAL to send the information to the hardware. To retrieve status, captured data and statistics the application uses a ‘get’ option which retrieves the information from IxHAL into IxTclHAL. A ‘cget’ option retrieves these values for use in Tcl applications.

Discussions of Tcl commands can be found in the following chapters:

- *IxTclHal API Description*: A discussion of the Tcl commands in IxTclHAL.
- *Appendix A - IxTclHAL Commands*: A complete description of the Tcl Command Library.
- *Appendix B - Utility Commands*: A number of additional provided utility commands.a number of additional Tcl commands that are used in most tests.
- *Appendix C - High-Level API*: A number of additional Tcl commands that are used in most tests.

Custom applications or test scripts can be written using Ixia's Tcl Command Library. For Windows users, as in standard Tcl/Tk packages, Ixia provides a Dynamic Link Library (DLL) file for Windows 2000/XP that may be loaded into a standard Tcl shell or Wish Console. The DLL gives access to the IxTclHal Command library.

For Unix users, the IxTclHal package connects to an instance of a TclServer on a Ixia chassis, where the DLL is used.

After installing the Tcl Client on the workstation, the Tcl package can be loaded by launching the Tcl Shell (double-clicking the *Wish Console* icon on the Desktop) and typing in the following command:

```
%package require IxTclHal
```

Now all the Ixia Tcl commands are available. If a new script is to be written, this should be the first line of the script file. The package command can also be used inside a previously written script, which could be loading other Tcl extensions such as Expect, Tcl-DP.

ScriptGen

ScriptGen is an auxiliary Tcl tool that is installed as part of the Tcl Client package. Its purpose is to create a Tcl program which reflects the configuration of a particular port. ScriptGen is run from a Wish Console and the resulting program is written to disk and shown in the console window. The configuration of the port may have been established through the use of any of the following Ixia tools: IxExplorer, IxAutomate (formerly IxScriptmate), or TCL API. The operation of ScriptGen is described in Appendix A of the *IxExplorer User Guide*.

What's New in Version 6.80 EA?

This section discusses the new TCL API features implemented in the 6.80 EA release.

- [TCLTransceiver](#) on page A-794
- [*chassis addVirtualCard chassisIP cardIP cardId keepAliveTimeout*](#) on page A-103
 - Added the following chassisChain StarTopology sub-commands.
 - chassisChain enableStarTopology
 - chassisChain disableStarTopology
 - chassisChain isValidForStarTopology
 - chassisChain isStarTopology
- See [*chassisChain enableStarTopology*](#) on page A-108.
- Added cfp4Qsfp28Type Transceiver

All Deprecated Commands and Options

The following table lists the commands, sub-commands and options which have been deprecated through the lifetime of the IxOS Tcl API. Refer to the appropriate guide release to determine the reason for the deprecation.

Note that the Usability column displays whether the command is a placeholder or it should not be used. The explanation of these options is as follows:

- Placeholder: The command or option has been coded to return a set value so as not to break previous scripts.
- Do not use: The command can still be used, but we do not recommend as it might break scripts in future releases.

Table 1-1. All deprecated commands and options

Command	Sub-commands	Options	First deprecated release	Usability
<many commands>	decode - chassis, card, port are required		3.70	Placeholder
atmHeader		ATM encapsulation options that do not begin with <i>atmEncapsulation</i>	3.80	Placeholder
calculateFPS calculateFrameRate	All	All	5.10	Do not use
calculateGapBytes	All	All	3.80	Do not use
captureFilterError		errOversize, errUndersize, errFragment	5.10	Do not use
card	getInterface		3.70	Do not use
		(type) cardUSB	5.20	Do not use
		txClockDeviationLan	5.30SP1	Do not use
		txClockDeviationWan	5.30SP1	Do not use
chassis		baseAddressMask	5.00	Do not use
chassis	write		4.10	Do not use
	writeAll		4.10	Do not use
	addVMCard		6.80	Do not use
filterPallette		type1, type2 typeMask1, typeMask2	3.65	Do not use

Table 1-1. All deprecated commands and options

Command	Sub-commands	Options	First deprecated release	Usability
filterPallette	config	-pattern1 "080201' used to work, but no longer. In 5.0 and up, must be specified as filterPallette config -pattern1 '08 02 01'.	5.0	Use space between hex bytes
frameRelay	dliciCore		4.10	Do not use
interfaceEntry		atmMode ATM encapsulation options that do not begin with <i>atmEncapsulation</i>	3.80	Placeholder
interfaceTable	all	sendArpClear [<i>description</i>] (only the description part of the command is deprecated)	5.00	Do not use
ipAddressTableItem		ATM encapsulation options that do not begin with <i>atmEncapsulation</i>	3.80	Do not use
isl		encapSA, encapDA hsa	3.80 4.10	Do not use Do not use
ixInitialize	All	All	3.80	Do not use
ixIs__Installed	All	All	3.80	Do not use
licenseManagement			5.00	Do not use
port		dataScrambling lineScrambling portFeaturePacketFlow-ImageFile portPosFraming portEthernetFraming rateMode sonetInterface sonetOperation useRecoveredClock (portMode) portUsbMode (type) portUsbUsb portUsbEthernet port10100UsbSh4	3.70 4.10 pre 3.65 3.65 3.70 3.70 3.65 5.20 5.20	Do not use Do not use
	getInterface		3.70	Placeholder

Table 1-1. All deprecated commands and options

Command	Sub-commands	Options	First deprecated release	Usability
portGroup	get		pre 3.65	Do not use
protocol		dutStripTag	3.65	Placeholder
protocoloffset	enable		4.10	Placeholder
sonet	B1, B2, B3		3.50	Do not use
	errorDuration		3.50	Do not use
	insertBipErrors		3.50	Do not use
	lossOfFrame		3.50	Do not use
	lossOfSignal		3.50	Do not use
	periodicB1, B2, B3		3.50	Do not use
	periodicLossOfFrame		3.50	Do not use
stat	enableUsbExtendedStats		4.00	Placeholder
		usb*	4.00	Placeholder
		counterVal	5.10	Do not use
		counterRate	5.10	Do not use
stream	fir		3.70	Do not use
	fcs options: good, alignErr, dribbleErr, bad, none		3.80	Do not use
	rateMode:useGap, usePercentRate		3.80	useGap deprecated in favor of ifg; usePercentRate deprecated in favor of percentPacketRate
	setGaps		5.10	Do not use
	setIFG		5.10	Do not use
	setLoopCount		5.20	Do not use
	setNumFrames		5.20	Do not use
streamQueue		aal5BitRate	3.80	Do not use
tcp		options	5.10	Do not use
timeserver		timeSource (some options)	4.00	Do not use
		e1T1Status	4.00	Do not use

Table 1-1. All deprecated commands and options

Command	Sub-commands	Options	First deprecated release	Usability
udf		timeOfDay	4.00	Do not use
		counterType	5.10	on boards and modes that support <i>udfSize</i> , <i>countertype</i> is deprecated in favor of <i>udfSize</i>
usb	Entire command	Entire command	4.00	Placeholder
weightedRandomFrameSize		randomType: UUNet and Lucent options	3.80	Do not use
fcoeProperties		enableKeepAlives	5.70	Do not use

2

Quick Start

Installing the IxOS Tcl Client

This chapter provides a quick means of getting started with the Tcl API. An example test is presented and explained.

The IxOS Tcl Client provides an interface between an Ixia Tcl client application and Ixia IxOS Tcl functions. It runs on the Unix / Linux host.

The Windows version of IxOS Tcl Client is included with the IxOS software package; the Unix/Linux version is supplied as a separate a self-extracting archive (.bin) file. You can download it from Ixia's website, www.ixiacom.com.

There are several versions of the IxOS Tcl Client. The correct file to install depends on the set up of the UNIX/Linux machine. [Table 2-2](#) on page 2-1 details the files and their use.

Table 2-2. Tcl Client Install Files

Install File	Purpose
IxOS# ## genericLinux.bin	For Linux versions post Redhat 9. It is distributed as a tarball (IxOS# ## genericLinux.bin.tar.gz) due to download issues.
IxOS# ## linux.bin.	For Linux platforms older than Redhat 9.
IxOS# ## setup.jar	An installer without a bundled Java Virtual Machine. This is distributed only to customers that have issues running the bin installers. It requires a Java Virtual Machine installed on the installation target.
IxOS# ## solarisSparc.bin	For Solaris machines.

The versions of UNIX/Linux operating systems that are supported are:

- Mandrake 7.2, RedHat 6.2, RedHat 7.0, RedHat 9.0
- RedHat Enterprise 4.0

- Solaris 2.7 (7), 2.8 (8), 2.9 (9)

Other versions of Linux and Solaris platforms may operate properly, but are not officially supported.

To install the IxOS TCL Client, do the following:

1. Download the self-extracting archive that contains the Unix/Linux Tcl client.
2. Use the following command to make the archive file executable:

```
chmod +x <archive file name>
```

Example: chmod +x IxOS#.##genericLinux.bin

(where #.## is the version number)

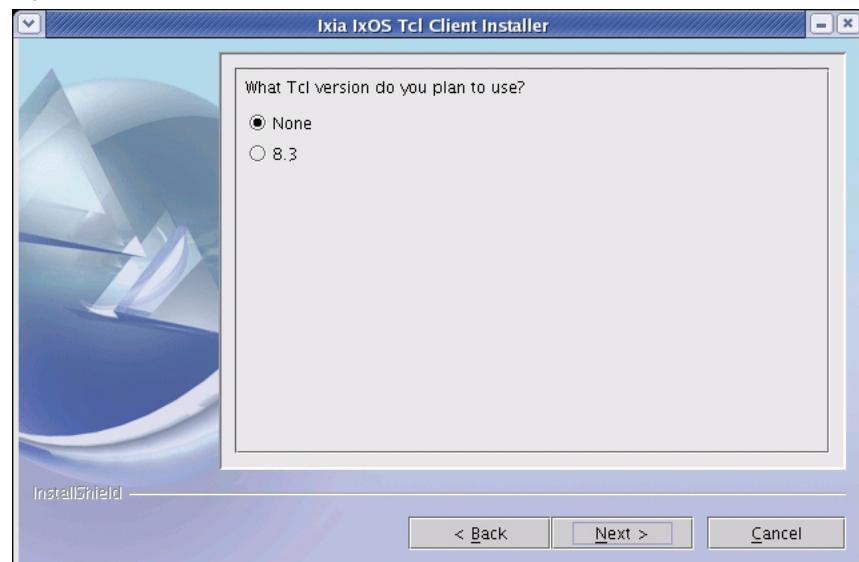
3. Execute the archive to extract the installation files and begin the installation:

```
./<archive file name>
```

Example: ./IxOS#.##genericLinux.bin

The installation is a typical InstallShield installation. The installer prompts you to select which version of the Tcl Client you want to install ([Figure 2-2](#)).

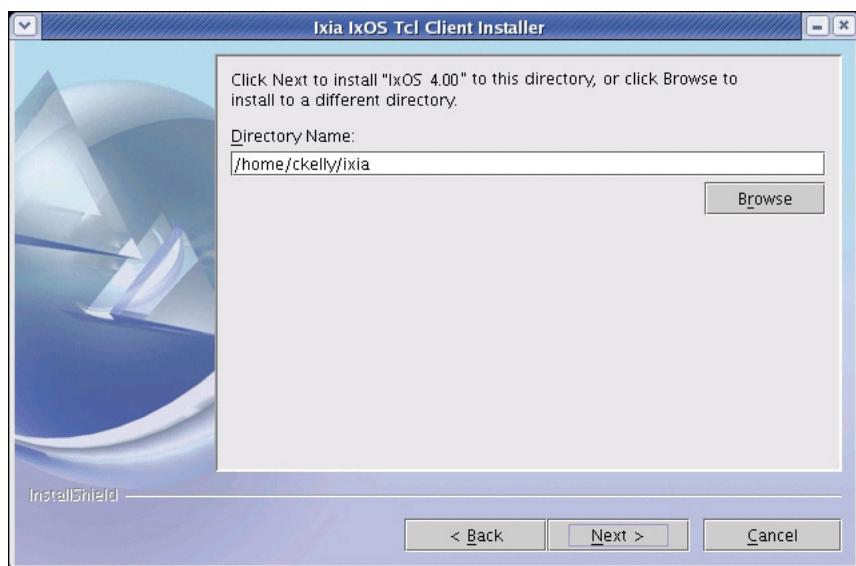
Figure 2-2. Tcl Version selection



The installer also prompts you to select the path where the Tcl Client is installed. ([Figure 2-3](#)). The default path is the current folder.

4. Accept the default installation path or enter an alternative, then click Next.

Figure 2-3. Installation path



UNIX Environment

On UNIX system, when IxOS client is installed, an ixwish file is created under “bin dir”. This ixwish file sets up the environment variables needed to run the IxOS client, and starts the wish shell.

The contents of the file are shown below. The highlighted lines are the environment variables that are set according to the user’s installation. To use Tcl libraries not supplied by IxOS, you should change the TCL_HOME and TCLver variables to point to the correct libraries.

```
#!/bin/sh
#
# Environment variables here are all caps if saved and exported
# Upper and lower case if used temporarily to set other
# variables

IXIA_HOME=<install location>

# USER MAY NEED TO CHANGE THESE IF USING OWN TCL LIBRARY

TCL_HOME=<install location>
TCLver=8.5

# USER NORMALLY DOES NOT CHANGE ANY LINES BELOW

IxiaLibPath=$IXIA_HOME/lib
IxiaBinPath=$IXIA_HOME/bin
TCLLibPath=$TCL_HOME/lib
TCLBinPath=$TCL_HOME/bin
TCL_LIBRARY=$TCLLibPath/tcl$TCLver
TK_LIBRARY=$TCLLibPath/tk$TCLver
PATH=$IxiaBinPath:$TCLBinPath:$PATH
#TCLLIBPATH=$IxiaLibPath:$TCLLIBPath;
# does not work, not a set of paths, must point to Ixia only
TCLLIBPATH=$IxiaLibPath
LD_LIBRARY_PATH=$IxiaLibPath:$TCLLibPath:$LD_LIBRARY_PATH
IXIA_RESULTS_DIR=/tmp/Ixia/Results
IXIA_LOGS_DIR=/tmp/Ixia/Logs
IXIA_TCL_DIR=$IxiaLibPath
IXIA_SAMPLES=$IxiaLibPath/ixTcl1.0
IXIA_VERSION=<build version>
export IXIA_HOME TCL_LIBRARY TK_LIBRARY TCLLIBPATH
export LD_LIBRARY_PATH IXIA_RESULTS_DIR
export IXIA_LOGS_DIR IXIA_TCL_DIR IXIA_SAMPLES
export IXIA_TCL_DIR PATH IXIA_VERSION
$TCLBinPath/wish ${@+"$@"}
```

Windows Environment

On Windows operating system, the IxiaWish.tcl file is installed as part of the IxOS client installation. The path to the IxiaWish.tcl file is similar to this:
C:\Program Files\Ixia\IxOS\<version>\TclScripts\bin\IxiaWish.tcl

This IxiaWish.tcl file sets up the environment variables needed to run the IxOS Tcl client. When Tcl 8.5 wish is started, IxiaWish.tcl is sourced to set up the IxTclHAL environment as part of the startup.

Alternatively, you can use a third-party wish like ActiveTcl, and set up the environment for accessing the IxOS Tcl package by sourcing the file named in the path above.

Installing Platform Independent package - tarball

To install *tarball*, perform the following steps:

- 1.** Create a folder *tarball* inside the root directory.
- 2.** Copy the *IxOSTclClient.tar.gz* inside the tarball and then extract it as *tar -zxvf IxOSTclClient.tar.gz*.
- 3.** Two folders *lib* and *TclDevelopmentGuide.pdf* are created.
- 4.** Create folders *tcl8.5* and *mpexpr-1.1* inside *lib*.
- 5.** Create another folder *activetcl* inside *tarball*.
- 6.** Download the tar file http://www.activestate.com/activetcl/downloads/thank-you?dl=http://downloads.activestate.com/ActiveTcl/releases/8.5.13.0/ActiveTcl8.5.13.0.296436-linux-x86_64-threaded.tar.gz and copy it to the *activetcl* folder.
- 7.** Extract the above tar file as *tar -zxvf ActiveTcl8.5.13.0.296436-linux-x86_64-threaded.tar*. A folder *ActiveTcl8.5.13.296436-linux-x86_64-threaded* gets created.
- 8.** Copy the contents of *tcl8.5* folder located in */root/tarball/activetcl/ActiveTcl8.5.13.296436-linux-x86_64-threaded/payload/lib* to *tcl8.5* folder located in */root/tarball/lib*.
- 9.** Copy the contents of the *mpexpr-1.1* folder located inside the IxOS installation directory */root/ixos1/lib* to *mpexpr-1.1* folder located inside */root/tarball/lib*.
- 10.** Set the environment file as:

```
# /bin/bash Environments:
# Source this file to setup environment variables
IXIA_HOME=/root/tarball
TCL_LIBRARY=/root/tarball/lib/tcl8.5
LD_LIBRARY_PATH=/root/tarball/
lib:$IxiaLibPath:$TclLibPath:$LD_LIBRARY_PATH
IXIA_TCL_DIR=/root/tarball/lib
TCLLIBPATH="$LOCAL_TCllibPATH /root/tarball/lib"
IXIA_SAMPLES=/root/tarball/lib/ixTcl1.0
IXIA_VERSION=6.60
export TCLLIBPATH IXIA_HOME TCL_LIBRARY LD_LIBRARY_PATH
export IXIA_SAMPLES IXIA_TCL_DIR IXIA_VERSION
```

Connect to IxServer

To connect to IxServer, perform the following steps:

- 1.** Source the environment file for tarball.
- 2.** *ixConnectToTclServer <chassis-name>*
- 3.** *ixConnectToChassis <chassis-name>*

IxSampleTcl Test Program

The IxSampleTcl.tcl file is included just below, along with comments which explain the test.

```
#####
#####
# IxTclHAL Version :5.20.0.165
# Product version :5.20.0 Build 165#
# File: IxSampleTCL.tcl
#
# Copyright © 1997 - 2009 by IXIA
# All Rights Reserved.
#
# The following is an example of how streams, ports and filters
# are configured,
# data capture started, transmission started and statistics
# collected.
# The chassis is connected to first, streams are created,
# filters are set,
# then capture is started on Rx port and transmisssion is
# started on Tx port.
# After the transmition is complete, some statistics are
# collected and
# displayed
# to standard out.
# Note: This test requires two ports which should be connected
# via loopback
# cable.
#
#####
#####
# This package is required to have access to the Ixia Tcl
# commands
package req IxTclHal
set userName IxiaTclUser
set hostname localhost
set retCode $::TCL_OK

# If on unix/linux, we must connect to the tclServer. This would
# need to be
# uncommented and a tclServer host name would need to be
# supplied. :
# NOTE: IxTclServer should not run on the chassis.
#if {[isUNIX]} {
# set retCode [ixConnectToTclServer $hostname]
#}
ixPuts "\n\tIxia Tcl Sample Script"
# Log in user
ixLogin $userName
ixPuts "\nUser logged in as: $userName"
set recCode [ixConnectToChassis $hostname]
if {$recCode != $::TCL_OK} {
    return $retCode
}

set chasId [ixGetChassisID $hostname]
set card 1
#added line below on July 2 to make ports selectable instead of
hardwired
set port1 3
```

```

set port2 4

# Assume transmit from port 1 to port 2 on same card for this
example
set portList [list [list $chassisId $card $port1] [list $chassisId
$card $port2]]
# Decide on some mac & ip addresses - lots of ways to do this,
this is one
# example

set macAddress(sa,$chassisId,$card,$port1) [format "be ef be ef
%02x %02x" $card $port1]
set macAddress(sa,$chassisId,$card,$port2) [format "be ef be ef
%02x %02x" $card $port2]
set macAddress(da,$chassisId,$card,$port1)
$macAddress(sa,$chassisId,$card,$port2)
set macAddress(da,$chassisId,$card,$port2)
$macAddress(sa,$chassisId,$card,$port1)

set ipAddress(sa,$chassisId,$card,$port1) [format "199.17.%d.%d"
$card $port1]
set ipAddress(sa,$chassisId,$card,$port2) [format "199.17.%d.%d"
$card $port2]
set ipAddress(da,$chassisId,$card,$port1)
$ipAddress(sa,$chassisId,$card,$port2)
set ipAddress(da,$chassisId,$card,$port2)
$ipAddress(sa,$chassisId,$card,$port1)

# Take ownership of the ports
if [ixTakeOwnership $portList] {
    return $::TCL_ERROR
}

proc clearOwnershipAndLogout {} {
    global portList
    ixClearOwnership $portList
    # Log off user
    ixLogout
    cleanUp
}

# Display version information
ixPuts "\nIxTclHAL Version :[version cget -ixTclHALVersion]"
ixPuts "Product version :[version cget -productVersion]"
ixPuts "Installed version :[version cget -installVersion]\n"
# Set ports to factory defaults. Dumps out on error.
ixPuts "Setting ports to factory defaults..."
foreach item $portList {
    scan $item "%d %d %d" chassisId card port
    if [port setFactoryDefaults $chassisId $card $port] {
        errorMsg "Error setting factory defaults on
        $chassisId.$card.$port]."
        clearOwnershipAndLogout
        return $::TCL_ERROR
    }
}
# Writes port properties in hardware
if {[ixWritePortsToHardware portList]} {
    clearOwnershipAndLogout
    return $::TCL_ERROR
}

# Check the link state of the ports

```

```

if {[ixCheckLinkState portList]} {
    clearOwnershipAndLogout
    return $::TCL_ERROR
}

ixPuts "Configuring streams..."
ixGlobalSetDefault
protocol config -ethernetType ethernetII
protocol config -name ip
# Set up some generic stream config items that are shared on all
streams
# generated.

stream config -numFrames 10
stream config -rateMode streamRateModePercentRate
stream config -percentPacketRate 42

foreach item $portList {
    scan $item "%d %d %d" chasId card port
    set frameSize 64 ;# we will make 20 streams w/incre.
framesizes

    ip config -sourceIpAddr $ipAddress(sa,$chasId,$card,$port)
    ip config -destIpAddr $ipAddress(da,$chasId,$card,$port)

    #####debug lines added by cz june 26#####
    puts "debug info source IP address for port $port
is:$ipAddress(sa,$chasId,$card,$port)"
    puts "debug info destination IP address for port $port
is:$ipAddress(da,$chasId,$card,$port)"
    #####debug#####

    if [ip set $chasId $card $port] {
logMsg "Error setting IP on $chasId,$card,$port"
set retCode $::TCL_ERROR
break
    }
    stream config -sa $macAddress(sa,$chasId,$card,$port)
    stream config -da $macAddress(da,$chasId,$card,$port)

    #####debug lines added by cz june 26#####
    puts "debug info source MAC for port $port
is:$macAddress(sa,$chasId,$card,$port)"
    puts "debug info destination MAC for port $port
is:$macAddress(da,$chasId,$card,$port)"
    #####debug#####

    stream config -dma advance
    set udfPattern [lrange [stream cget -da] 2 end]
    #####debug lines added by cz june 26#####
    puts "udfPattern is :$udfPattern"
    #####debug#####
    udf config -enable true
    udf config -offset 42
    udf config -initval $udfPattern
    udf config -countertype c32
    udf config -maskselect {00 00 00 00}
    udf config -maskval {00 00 00 00}
    udf config -random false
}

```

```

        udf config -continuousCount false
        udf config -repeat 1
        if [udf set 4] {
            errorMsg "Error setting UDF 4"
            set retCode $::TCL_ERROR
            break
        }
        # Configure 20 streams on Tx port
        for {set streamId 1} {$streamId < 20} {incr streamId} {
            stream config -name "Stream $streamId - IP sample stream"
            stream config -framesize $frameSize
            incr frameSize 42
            if [stream set $chassisId $card $port $streamId] {
                errorMsg "Error setting stream $chassisId,$card,$port.$streamId"
            } else {
                set retCode $::TCL_ERROR
                break
            }
        }

        incr streamId -1
        # Set last stream to STOP
        stream config -dma stopStream
        if [stream set $chassisId $card $port $streamId] {
            errorMsg "Error setting stream $chassisId,$card,$port.$streamId"
        } else {
            set retCode $::TCL_ERROR
            break
        }

        ##### TK change
        set rxUdfPattern [lrange $macAddress (sa,$chassisId,$card,$port) 2 end]

        # Set the filter parameters
        filterPallette config -pattern2 $rxUdfPattern
        filterPallette config -patternOffset2 [udf cget -offset]
        filter config -userDefinedStat2Pattern pattern2
        filter config -userDefinedStat2Enable true
        filter config -userDefinedStat2Error errGoodFrame
        filter config -captureTriggerEnable true
        filter config -captureFilterEnable true
        if [filterPallette set $chassisId $card $port] {
            errorMsg "Error setting filter pallette for
$chassisId,$card,$port."
            set retCode $::TCL_ERROR
            break
        }
        if [filter set $chassisId $card $port] {
            errorMsg "Error setting filters on $chassisId,$card,$port."
            set retCode $::TCL_ERROR
            break
        }
    }

    # Dump out now if there were any errors.. maybe you want to throw
    instead of a
    # return.
    if {$retCode != $::TCL_OK} {
        clearOwnershipAndLogout
        return $retCode
    }
}

```

```

# Writes all the configuration on ports in hardware
# NOTE: This does NOT take link down, so no point in checking
link state
# afterward and no need for any delays
# Also note that this is an example of a throw instead of a
return
if [ixWriteConfigToHardware portList] {
    return -code error
}
# Zero all statistic counters on ports
if [ixClearStats portList] {
    return -code error
}
ixPuts "Start capture..."
if [ixStartCapture portList] {
    return -code error
}
ixPuts "Start transmit..."
if [ixStartTransmit portList] {
    return -code error
}
# Let it transmit for a bit; if this were a real test, we might
want to wait for
# approx. the total transmit time. Since it's not, 1 sec is
sufficient for the
# streams we've created.
after 1000
# Checks whether transmission is done on a group of ports
if {[ixCheckTransmitDone portList] == $::TCL_ERROR} {
    clearOwnershipAndLogout
    return -code error
} else {
    ixPuts "Transmission is complete."
}

# Stop capture on ports - not really necessary, as any read of
capture will
# automatically stop capture
ixPuts "Stop capture..."
if [ixStopCapture portList] {
    clearOwnershipAndLogout
    return -code error
}

# This api will request stats from all ports in the portList -
it's really
# efficient and the best way to collect stats when you have
multiple ports to
# contend with.
ixRequestStats portList
foreach item $portList {
    scan $item "%d %d %d" chasId card port
    if {[statList get $chasId $card $port]} {
        ixPuts "Error getting stats for $chasId,$card,$port"
        set retCode $TCL_ERROR
        break
    }
    # note that if a stat is not supported on a particular port
    type, the cget
    # will throw so it is best to protect that in the following
    fashion:

```

```
        if [catch {statList cget -scheduledFramesSent} numTxFrames ]  
        {  
        set numTxFrames 0  
        ixPuts "WARNING: -scheduledFramesSent not supported on  
$chassisId,$card,$port. Value set to 0"  
        }  
  
        if [catch {statList cget -userDefinedStat2} numRxFrames ] {  
set numRxFrames 0  
ixPuts "WARNING: -userDefinedStat2 not supported on  
$chassisId,$card,$port.  
Value set to 0"  
        }  
  
        if [captureBuffer get $chassisId $card $port 1 $numRxFrames] {  
ixPuts "Error getting captureBuffer on $chassisId $card  
$numRxFrames"  
set retCode $::TCL_ERROR  
#break removed by cz on July 2nd 2009  
#break  
        }  
  
        ixPuts "Port: $chassisId,$card,$port"  
        ixPuts -nonewline "Speed: [stat getLineSpeed $chassisId $card  
$port]\t"  
        ixPuts -nonewline "Frames sent: $numTxFrames\t"  
        ixPuts -nonewline "Frames Rcvd: $numRxFrames\t"  
        ixPuts "Number of packets captured :[captureBuffer cget -  
numFrames]\n"  
        }  
  
        ixPuts "\nSample test complete.\n"  
clearOwnershipAndLogout
```

3

High-Level and Utility API Description

This chapter presents a description of the High-Level API commands organized by major topics, as mentioned in the following list:

- *Initialization, Setup and Cleanup*—basic overhead to set up the test.
- *Port Ownership*—commands to control port ownership and sharing.
- *Data Transmission*—setup for data transmission.
- *Data Capture and Statistics*—setup for data capture and statistics.
- *Console Output and Logging*—output messages to the console and log files.

This chapter provides an overview of the high-level API functions and utility commands. The full details of the commands described herein may be found in the following appendices:

- *Appendix B - Utility Commands* includes complete descriptions of each of the Utility commands.
- *Appendix C - High-Level API* includes complete descriptions of each of the high-level commands.

The high-level commands are characterized by one or more of the following characteristics:

- They perform a combination of IxTclHAL commands.
- They perform one or more IxTclHAL commands over a range of ports.
- They control test operation sequences.

Arguments to the high-level APIs are passed in one of the following two ways:

- **By value:** Denoted by (By value) in the Appendix C description. By value arguments are either a constant or a \$variable reference. For example:
`32, {{1 1 1} {1 2 1}} or $portList`
- **By reference:** Denoted by (By reference) in the Appendix C description. By reference arguments must be references to variables, **without** the '\$'. For example, `pl` after `set pl {{1 1 1} [1 1 2]}` or `one2oneArray`.

Read the individual description pages in the Appendices to determine which arguments are passed by reference and by value.

Initialization, Setup and Cleanup

The commands in this section relate to overhead operations necessary to prepare for data transmission, capture and statistical analysis. The commands covered in this section are the following:

- *Mapping and Port Lists*
 - *map*
 - *ixCreatePortListWildCard*
 - *ixCreateSortedPortList*
 - *getAllPorts, getRxPorts and getTxPorts*
- *Including Source Code*
 - *ixSource*
- *Chassis and TclServer Connection*
 - *ixConnectToTclServer / ixDisconnectTclServer*
 - *ixProxyConnect*
 - *ixConnectToChassis / ixDisconnectFromChassis*
 - *ixGetChassisID*
 - *user*
- *General Purpose Commands*
 - *ixWritePortsToHardware*
 - *ixWriteConfigToHardware*
- *cleanUp*
 - *cleanUp*

Mapping and Port Lists

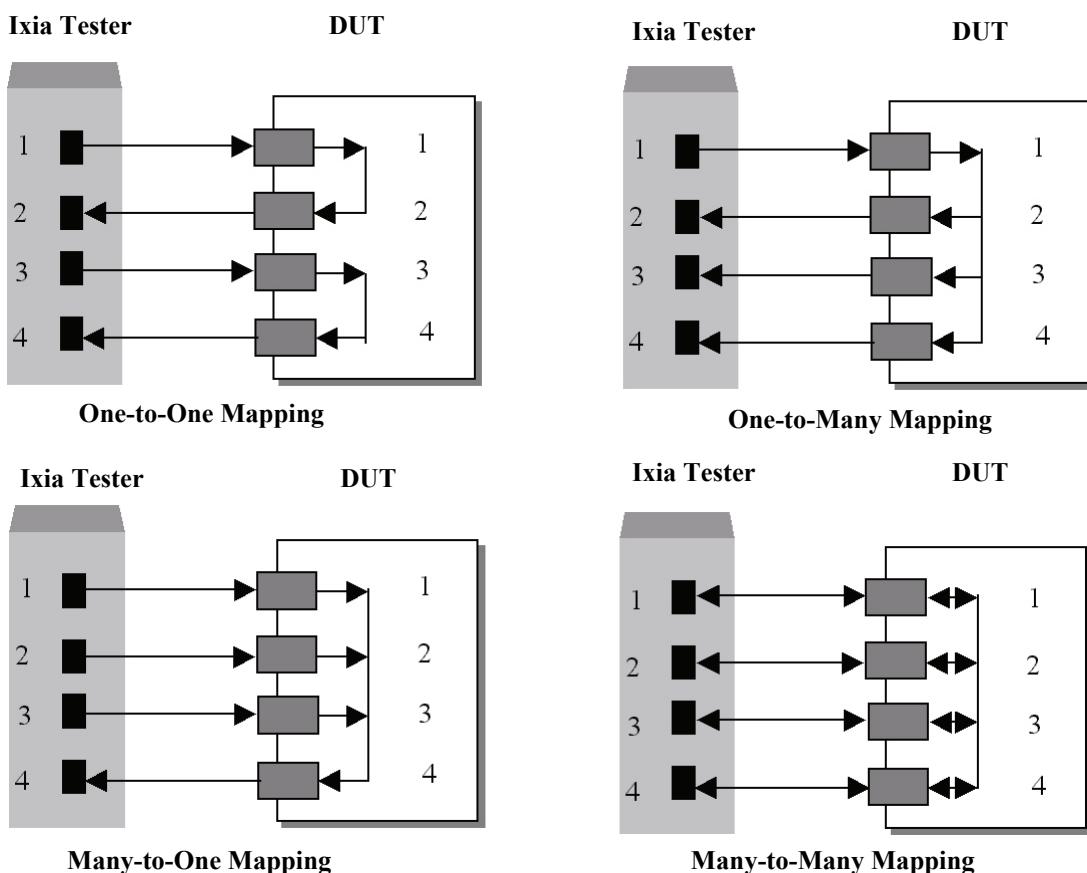
Four types of traffic mappings are common in TCL tests, as shown in *Figure 3-1 on page 4*.

1. *One-to-one mapping*: One transmit port is mapped to one receive port. For example, port 1 of the Ixia chassis transmits to port 1 of the DUT, which forwards traffic back on its port 2 to Ixia chassis port 2.
2. *One-to-many mapping*: One transmit port is mapped to multiple receive ports. For example, port 1 of the Ixia chassis transmits to port 1 of the DUT, which forwards back on its ports 2, 3, and 4 to Ixia chassis ports 2, 3, and 4.
3. *Many-to-one mapping*: Multiple transmit ports mapped to a single receive port. For example, ports 1, 2, and 3 of Ixia chassis transmit to ports 1, 2, and 3 of the DUT, which forwards back on its port 4 to Ixia chassis port 4.
4. *Many-to-many mapping*: Multiple transmit ports are mapped to multiple receive ports. For example, port 1 of the Ixia chassis transmits to port 1 of the DUT, which forwards back on its ports 2, 3, and 4 to Ixia chassis ports 2, 3, and 4; at the same time, port 2 of the Ixia chassis transmits to port 2 of the

DUT which forwards back on its ports 1, 3, 4 to Ixia chassis ports 1, 3, and 4; and so on. In this mapping, all ports transmit to and receive from all other ports in the system.

The traffic mapping is a logical collection of ports and configurations stored in memory. It simplifies the identification of transmit and receive ports during the configuration of streams and filters.

Figure 3-1. Traffic Mappings



Tcl programmers find it convenient to configure their ports using the `map` utility command in one of the following four global arrays:

- *one2oneArray*: Sets up one transmit and one receive port for traffic flow. The transmit/receive port pair that has been configured once cannot be used in a different port pair. That is, each port pair is mutually exclusive.
- *one2manyArray*: Sets up one transmit port and multiple receive ports. Each group of transmit and its multiple receive ports is mutually exclusive with other groups.
- *many2oneArray*: Sets up multiple transmit ports and one receive port. Each group of multiple transmit ports and its receive port is mutually exclusive with other groups.

- *many2manyArray*: Sets up multiple transmit ports and multiple receive ports. Any port may transmit and receive to any other port in any group of ports.

map

The map command is used to define any of the four basic array types. Refer to “*map*” on page C-191 for full details. The important options and sub-commands of this command are:

Table 3-1. map Options

Member	Usage
type	The type of mapping; one of <i>one2one</i> , <i>one2many</i> , <i>many2one</i> or <i>many2many</i> .

Table 3-2. map Sub-Commands

Member	Usage
add	Adds a new transmit port - receive port pair to the mapping.
del	Deletes a transmit port - receive port pair from the mapping.
new	Clears the current map and creates a new map as described in type, above.
show	Shows the current map configuration.

Any of the four global arrays may be used in most of the high-level commands where a *portList* is called for as mentioned in the following list:

- *one2oneArray*
- *one2manyArray*
- *many2oneArray*
- *many2manyArray*

The command uses the part of the array appropriate to the command; for example, *ixStartTransmit* uses only the transmit ports of the array. Alternatively, any command that calls for a *portList* may construct an array of ports and use it as an argument. Two alternative forms are defined in the following list:

- `{{{1 1 1} {1 1 2} {1 1 3} {1 1 4}}}`: four ports: chassis 1, card 1, ports 1-4
- `{1,1,1 1,1,2 1,1,3 1,1,4}`: four ports: chassis 1, card 1, ports 1-4. Specifications are separated by spaces.

ixCreatePortListWildCard

Port lists may of course be created by hand. For example:

```
{ {1 1 1} {1 1 2} {1 1 3} {1 1 4} }
```

The `ixCreatePortListWildCard` command can be used to build a sorted list containing wild card characters (*) to indicate all cards and/or all ports. For example,

```
ixCreatePortListWildCard {{1 * *}} - all cards and all
ports on chassis 1

ixCreatePortListWildCard {{1 1 *} {1 2 1} {1 2 2}} - all
ports on card 1 and ports 1 and 2 on card 2.
```

A wild card cannot be used for chassis ID. Also, if a combination of a list element containing wild cards and port numbers are used, then the port list passed **must** be in a sorted order. The format of this command is as follows:

ixCreatePortListWildCardOptions *portList* [*excludePortList*]

where *portList* is the list of ports (with wildcards) to be included, and *excludePortList* is a list of ports, which may not contain wildcards, which should be omitted from the returned list.

Refer to “*ixCreatePortListWildCard*” on page 49 for full details of this command.

ixCreateSortedPortList

The `ixCreateSortedPortList` command can be used to construct a port list for a range of ports—from a port on a single card to another port on a different card. For example:

```
ixCreateSortedPortList {1 1 1} {1 5 4} {{1 3 2}} - all
ports between chassis 1 card 1 port 1 and port 4 on card
5, excluding card 3 port 2.
```

The format of this command is as follows:

ixCreateSortedPortList *portFrom* *portTo* *exclude*

where *portFrom* is the first port in the range and *portTo* is the last port in the range. These are individual port specifications—not a list of lists as in other commands. *exclude* is a list of lists indicating individual ports to be omitted from the list; an empty list is expressed as {}.

Refer to “*ixCreateSortedPortList*” on page C-51 for full details of this command.

getAllPorts, getRxPorts and getTxPorts

These three utility command all serve to retrieve the ports associated with a map array. The three commands are the following:

- “*getAllPorts*” on page C-2: Returns all of the ports associated with an array.
- “*getRxPorts*” on page C-3: Returns just the receive ports associated with an array.

- “*getTxPorts*” on page C-4: Returns just the transmit ports associated with an array.

Including Source Code

ixSource

The **ixSource** command is very useful in sourcing large number of .tcl files from a folder or a number of individual files. It may be called with either a single folder name or a set of full path names. In the former case, all the .tcl files within the folder are sourced and in the latter case, each of the individual files are sourced. The format of this command is as follows:

ixSource {fileNames | directoryName}

where *fileNames* is any number of files to be sourced and *directoryName* is the folder name where all the files under that folder are going to be sourced.

Refer to “*ixSource*” on page C-136 for full details on this command.

Chassis and TclServer Connection

Several commands are available to initialize connection to the chassis chain to be used for testing. Provisions are included to connect to TclServer from Unix platforms.

ixConnectToTclServer / ixDisconnectTclServer

It can be used from a Unix client to connect to a host that runs the TclServer, or disconnect from the server. The format of this command is as follows:

ixConnectToTclServer *serverName*

ixDisconnectTclServer

where *serverName* is the hostname or IP address of the Windows based machine hosting the IxTclServer.

Refer to “*ixConnectToTclServer*” on page C-47 and “*ixDisconnectTclServer*” on page C-57 for a full description of this command.

ixProxyConnect

The **ixProxyConnect** command combines the functions of **ixTclSrvConnect** and **IxConnectToChassis**. The format of this command is as follows:

ixProxyConnect *chassisName* *chassisList* [*cableLen* [*logfilename*]]

where *chassisName* is the hostname or IP address of a host running TclServer which is used from Unix clients, *chassisList* is a list of all of the chassis in the chain - either IP addresses or host names that can be resolved through DNS, *cableLen* is the length of cables that connects the chassis, and *logfilename* is the file to create to store log messages.

Refer to “*ixProxyConnect*” on page C-86 for a full description of this command.

ixConnectToChassis / ixDisconnectFromChassis

The `ixConnectToChassis` command is called from `IxConnectToChassis`. It connects to a list of chassis given the host names or IP addresses. The format of these commands are as follows:

ixConnectToChassis *chassisList [cableLen]*

ixDisconnectFromChassis

where *chassisList* is a list of all of the chassis in the chain - either IP addresses or host names that can be resolved through DNS and *cableLen* is the length of cables that connects the chassis.

Refer to “*ixConnectToChassis*” on page C-44 and “*ixDisconnectFromChassis*” on page C-56 for a full description of these commands.

ixGetChassisID

This command obtains the chassis ID of a chassis given its hostname or IP address. This command is needed after using `ixConnectToChassis` or `ixProxyConnect` to obtain automatically assigned chassis IDs. The format of the command is:

ixGetChassisID *chassisName*

where *chassisName* is the hostname or IP address of the chassis. Refer to “*ixGetChassisID*” on page C-65 for a full description of this command.

user

This command has no effect on test operation. Rather it provides a means of storing global information about the user and the DUT. The only sub-commands available are `config` and `cget`. The important options of this command are mentioned in the following table:

Table 3-3. user Options

Member	Usage
productname	Name of the DUT being tested.
version	Version number of the product.
serial#	Serial number of the product.
username	The name of the user running the tests.

Refer to “*user*” on page B-24 for a full description of this command.

General Purpose Commands

The following two commands are invaluable tools for committing large amounts of configuration information to the hardware.

ixWriteConfigToHardware

This command commits the configuration of streams, filters, and protocol information on a group of ports to hardware. The format of the command is as follows:

ixWriteConfigToHardware *portList*

where *portList* is a list of ports to apply the command to. Refer to “*ixWriteConfigToHardware*” on page C-188 for full description of this command.

ixWritePortsToHardware

In addition to performing all of the functions of *IxWriteConfigToHardware*, this command commits the configuration of ports such as MII properties on 10/100 interface (speed, duplex modes, auto-negotiation), port properties on Gigabit interfaces, and PPP parameters on Packet over SONET interfaces on a group of ports to hardware. Link may drop as a result of this command’s execution. The format of the command is as follows:

ixWritePortsToHardware *portList*

where *portList* is a list of ports to apply the command to. Refer to “*ixWritePortsToHardware*” on page C-190 for full description of this command.

cleanUp

cleanUp

The *cleanUp* command may be used to reset the Ixia hardware and to undo the effects of the *package require IxTclHal* command. The state of the wish shell is reset back to its initial state with respect to Ixia software execution. Refer to “*.cleanUp*” on page B-9 for full description of this command.

Port Ownership

Ports on chassis may be shared among a number of users. The following commands in this section control user login and port sharing:

- *ixLogin / ixLogout*
- *ixCheckOwnership*
- *ixPortTakeOwnership / ixTakeOwnership / ixPortClearOwnership / ixClearOwnership*

An additional utility command is available to clear all port ownership for the current user. This is as follows:

- *clearAllMyOwnership*

ixLogin / ixLogout

The **ixLogin** command registers a name to associate with port ownership and the **ixLogout** command dissociates ownership. The format of these commands are as follows:

```
ixLogin ixiaUser
ixLogout
```

where *ixiaUser* is the name of the current user.

Refer to “**ixLogin**” on page C-78 and “**ixLogout**” on page C-79 for full details on these commands.

ixCheckOwnership

The **ixCheckOwnership** command is used to check for the availability of a number of ports before taking ownership. The format of this command is as follows:

```
ixCheckOwnership portList
```

where *portList* is a list of ports, which may contain wildcards. The **ixCheckOwnership** command requires that the list be passed by value. For example,

```
set p1 {{1 1 1} {1 1 2}}
ixCheckOwnership $p1
```

A value of 0 is returned if all of the ports are available. Refer to “**ixCheckOwnership**” on page C-13 for a full description of this command.

ixPortTakeOwnership / ixTakeOwnership / ixPortClearOwnership / ixClearOwnership

The **ixPortTakeOwnership** and **ixTakeOwnership** commands take ownership of a single port or list of ports, respectively. The **ixPortClearOwnership** and **ixClearOwnership** commands give the ports back. The format of these commands are as follows:

```
ixPortTakeOwnership chassisID cardID portID [takeType]
ixTakeOwnership portList [takeType]
```

ixPortClearOwnership *chassisID cardID portID [takeType]*

ixClearOwnership *[portList [takeType]]*

where *chassisID*, *cardID* and *portID* define an individual port, *portList* is a list of ports and *takeType* may be *force* to force the taking or release of ownership regardless of ownership by another user. The port list must be passed by value as in the *ixCheckOwnership* command. A call to *ixClearOwnership* without any arguments clears all ports owned by the currently logged on user.

Refer to “*ixPortTakeOwnership*” on page C-84, “*ixTakeOwnership*” on page C-181, “*ixPortClearOwnership*” on page C-82 and “*ixClearOwnership*” on page C-24 for complete descriptions of these commands.

Data Transmission

The data transmission commands relate the preparation for, or the transmission of data to the DUT. Several utility commands, which calculate frequently used values, are detailed as well. The commands covered are as follows:

- *Setup*
 - *ixCheckLinkState*
 - *ixCheckPPPState*
 - *ixSetPortPacketFlowMode / ixSetPacketFlowMode*
 - *ixSetPortPacketStreamMode / ixSetPacketStreamMode*
 - *ixSetPortAdvancedStreamSchedulerMode / ixSetAdvancedStreamSchedulerMode*
 - *ixSetPortTcpRoundTripFlowMode / ixSetTcpRoundTripFlowMode*
 - *disableUdfs*
- *Negotiation*
 - *ixRestartPortAutoNegotiation / ixRestartAutoNegotiation*
 - *ixRestartPortPPPNegotiation / ixRestartPPPNegotiation*
- *Start Transmit*
 - *ixStartPortTransmit / ixStartTransmit / ixStopPortTransmit / ixStopTransmit*
 - *ixStartStaggeredTransmit*
 - *ixCheckPortTransmitDone / ixCheckTransmitDone*
 - *ixStartPortCollisions / ixStartCollisions / ixStopPortCollisions / ixStopCollisions*
 - *ixStartPortAtmOamTransmit / ixStartAtmOamTransmit / ixStopPortAtmOamTransmit / ixStopAtmOamTransmit*
 - *ixClearScheduledTransmitTime / ixSetScheduledTransmitTime*
 - *ixLoadPoePulse / ixLoadPortPoePulse*
- *Calculation Utilities*
 - *calculateMaxRate*
 - *host2addr*
 - *byte2IpAddr*
 - *dectohex*
 - *hextodec*

Setup

ixCheckLinkState

The `ixCheckLinkState` command checks the link state on a group of ports. This command should be called early in the script to ensure that all links are up before any traffic is transmitted to the DUT. The format of the command is as follows:

ixCheckLinkState *portList*

where *portList* is the set of ports to check. A success value of *0* is returned if all of the ports have link.

Refer to “*ixCheckLinkState*” on page C-11 for a complete explanation of this command.

ixCheckPPPState

The `ixCheckPPPState` command checks the state on a group of POS ports. This command should be called early in the script to ensure that all POS ports are up before any traffic is transmitted to the DUT. The format of the command is as follows:

ixCheckPPPState *portList*

where *portList* is the set of ports to check. A success value of *0* is returned if all of the ports have link.

Refer to “*ixCheckPPPState*” on page C-15 for a complete explanation of this command.

ixSetPortPacketFlowMode / ixSetPacketFlowMode

These commands set the mode of the indicated ports to flow mode as opposed to stream mode. The format of these commands are as follows:

ixSetPortPacketFlowMode *chassisID cardID portID [write]*

ixSetPacketFlowMode *portList [write]*

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports. The *write* argument commits the settings to the hardware immediately.

Refer to “*ixSetPortPacketFlowMode*” on page C-117 and “*ixSetPacketFlowMode*” on page C-105 for a complete explanation of these commands.

ixSetPortPacketStreamMode / ixSetPacketStreamMode

These commands set the mode of the indicated ports to stream mode as opposed to flow mode. The format of these commands are as follows:

ixSetPortPacketStreamMode *chassisID cardID portID [write]*

ixSetPacketStreamMode *portList* [*write*]

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports. The *write* argument commits the settings to the hardware immediately.

Refer to “*ixSetPortPacketStreamMode*” on page C-121 and “*ixSetPacketStreamMode*” on page C-109 for a complete explanation of these commands.

**ixSetPortAdvancedStreamSchedulerMode /
ixSetAdvancedStreamSchedulerMode**

These commands set the mode of the indicated ports to advanced stream scheduler mode. The format of these commands are as follows:

```
ixSetPortAdvancedStreamSchedulerMode chassisID cardID portID  

[write]  

ixSetAdvancedStreamSchedulerMode portList [write]
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports. The *write* argument commits the settings to the hardware immediately.

Refer to “*ixSetPortAdvancedStreamSchedulerMode*” on page C-111 and “*ixSetAdvancedStreamSchedulerMode*” on page C-98 for a complete explanation of these commands.

ixSetPortTcpRoundTripFlowMode / ixSetTcpRoundTripFlowMode

These commands set the mode of the indicated ports to TCP round trip flow mode as opposed to flow or stream mode. The format of these commands are as follows:

```
ixSetPortTcpRoundTripFlowMode chassisID cardID portID [write]  

ixSetTcpRoundTripFlowMode portList [write]
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports. The *write* argument commits the settings to the hardware immediately.

Refer to “*ixSetPortTcpRoundTripFlowMode*” on page C-125 and “*ixSetTcpRoundTripFlowMode*” on page C-130 for a complete explanation of these commands.

disableUdfs

The *disableUdfs* command disables one or more UDFs. The format of the command is as follows:

```
disableUdfs udfList
```

where *udfList* is a list of values in the range 1-4. For example, {1 2 3 4}. A call to *stream set* is needed to write these values to the hardware.

Refer to “*disableUdfs*” on page B-12 for a full description of this command.

Negotiation

ixRestartPortAutoNegotiation / ixRestartAutoNegotiation

These commands are used to restart auto-negotiation on a port or list of ports. The format of these commands are as follows:

```
ixRestartPortAutoNegotiation chassisID cardID portID  
ixRestartAutoNegotiation portList
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “*ixRestartAutoNegotiation*” on page C-94 and “*ixRestartPortAutoNegotiation*” on page C-95 for complete descriptions of these commands.

ixRestartPortPPPNegotiation / ixRestartPPPNegotiation

These commands are used to restart PPP negotiation on a port or list of ports. The format of these commands are as follows:

```
ixRestartPortPPPNegotiation chassisID cardID portID  
ixRestartPPPNegotiation portList
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “*ixRestartPortPPPAutoNegotiation*” on page C-96 and “*ixRestartPortPPPAutoNegotiation*” on page C-96 for complete descriptions of these commands.

Start Transmit

ixStartPortTransmit / ixStartTransmit / ixStopPortTransmit / ixStopTransmit

These commands are used to start and then stop transmission on a single port or a group of ports. The *ixStartCapture* or *ixStartPortCapture* should be used before these commands. The format of these commands are as follows:

```
ixStartPortTransmit chassisID cardID portID  
ixStartTransmit portList  
ixStopPortTransmit chassisID cardID portID  
ixStopTransmit portList
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “*ixStartPortTransmit*” on page C-154, “*ixStartTransmit*” on page C-158, “*ixStopPortTransmit*” on page C-177 and “*ixStopTransmit*” on page C-179 for complete descriptions of these commands.

ixStartStaggeredTransmit

This command performs the same function as *ixStartTransmit*, but staggers the time from one port’s start to the next by 25 - 30ms. The format of this command is as follows:

ixStartStaggeredTransmit *portList*

where *portList* identifies a number of ports to start staggered transmission on.

Refer to “*ixStartStaggeredTransmit*” on page C-156 for a complete descriptions of this command.

ixCheckPortTransmitDone / ixCheckTransmitDone

These commands poll a single port or list of ports to determine when all data has been transmitted to the DUT. This command does not return until transmission is complete on all the ports referenced. **Note:** These commands should be called no earlier than one second after starting transmit with *ixStartTransmit* or *ixStartPortTransmit*. We recommend that an *after 1000* statement be included after each start transmit. The format of these commands are as follows:

ixCheckPortTransmitDone *chassisID cardID portID*

ixCheckTransmitDone *portList*

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “*SixCheckPortTransmitDone*” on page C-17 and “*ixCheckTransmitDone*” on page C-19 for a complete explanation of these commands.

ixStartPortCollisions / ixStartCollisions / ixStopPortCollisions / ixStopCollisions

These commands are used to start and then stop generation of forced collisions on a single port or list of ports. The *forcedCollisions* command should be used before these commands to set up the parameters for collision generation. The format of these commands are as follows:

ixStartPortCollisions *chassisID cardID portID*

ixStartCollisions *portList*

ixStopPortCollisions *chassisID cardID portID*

ixStopCollisions *portList*

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “[ixStartPortCollisions](#)” on page C-149, “[ixStartCollisions](#)” on page C-140, “[ixStopPortCollisions](#)” on page C-172 and “[ixStopCollisions](#)” on page C-163 for complete descriptions of these commands.

**ixStartPortAtmOamTransmit / ixStartAtmOamTransmit /
ixStopPortAtmOamTransmit / ixStopAtmOamTransmit**

These commands are used to start and then stop ATM OAM message transmit on a single port or list of ports. The `atmOam` command should be used before these commands to set up the parameters for collision generation. The format of these commands are as follows:

```
ixStartPortAtmOamTransmit chassisID cardID portID  
ixStartAtmOamTransmit portList  
ixStopPortAtmOamTransmit chassisID cardID portID  
ixStopAtmOamTransmit portList
```

where `chassisID`, `cardID`, and `portID` identifies a single port and `portList` identifies a number of ports.

Refer to “[ixStartAtmOamTransmit](#)” on page C-137, “[ixStopPortAtmOamTransmit](#)” on page C-169, “[ixStartPortAtmOamTransmit](#)” on page C-146 and “[ixStopPortAtmOamTransmit](#)” on page C-169 for complete descriptions of these commands.

ixClearScheduledTransmitTime / ixSetScheduledTransmitTime

These commands are used to reset and set the transmit duration for ports that support that feature. Streams may be programmed for continuous transmit and these commands used to limit the overall test to a period of time.

Refer to [ixClearScheduledTransmitTime](#) on page C-36 and [ixSetScheduledTransmitTime](#) on page C-127, for complete descriptions of these commands.

ixLoadPoePulse / ixLoadPortPoePulse

These commands are used to send a pulse on Power over Ethernet modules.

Refer to [ixLoadPoePulse](#) on page C-76 and [ixLoadPortPoePulse](#) on page C-77, for complete descriptions of these commands.

Calculation Utilities

calculateMaxRate

The `calculateMaxRate` command calculates the maximum frame rate for a port, based on the frame size and preamble size. The format of the command is as follows:

```
calculateMaxRate chassis card port [frameSize preambleOrAtmEncap]
```

where *chassis*, *card*, *port*: a port of the type that you wish the maximum frame rate calculated for;

frameSize: The size of the frame (*default* = 64);

preambleOrAtmEncap: The size of the preamble, or the ATM encapsulation used for ATM cards. The values for ATM encapsulation may be found in the *encapsulation* option of the [atmHeader](#) command. (*default* = 8).

host2addr

This command converts an IP address in dotted notation to a list of hex bytes. The format of the command is as follows:

host2addr *IPaddr*

where *IPaddr* is the address in dotted notation. The result is a list of four hex byte values.

Refer to “*host2addr*” on page B-18 for a full description of this command.

byte2IpAddr

This command converts a list of four hex bytes into an IP address in dotted notation. The format of the command is as follows:

byte2IpAddr *hexIPaddr*

where *hexIPaddr* is the address as a list of four hex byte values. The result is a dotted notation.

Refer to “*byte2IpAddr*” on page B-2 for a full description of this command.

dectohex

This command converts a decimal number to hexadecimal notation. The format of the command is as follows:

dectohex *deignum*

where *deignum* is the decimal value. The result is in hexadecimal notation.

Refer to “*dectohex*” on page B-11 for a full description of this command.

hextodec

This command converts a hexadecimal number to decimal notation. The format of the command is as follows:

hextodec *hexnum*

where *hexnum* is the hexadecimal value. The result is in decimal notation.

Refer to “*hextodec*” on page B-17 for a full description of this command.

Data Capture and Statistics

The commands in this section relate to setup for data capture, initiating data capture and collection of statistics. Although this section follows the one on data transmission, all capture setup and initiation should be done **before** any data transmission is started. The commands covered in this section are as follows:

- *Setup*
 - *ixSetPortCaptureMode / ixSetCaptureMode*
 - *ixSetPortPacketGroupMode / ixSetPacketGroupMode*
 - *ixClearTimeStamp*
 - *ixClearPortStats / ixClearStats*
 - *ixClearPortPacketGroups / ixClearPacketGroups*
 - *ixResetSequenceIndex / ixResetPortSequenceIndex*
- *Capture Data*
 - *ixStartPortCapture / ixStartCapture / ixStopPortCapture / ixStopCapture*
 - *ixStartPortPacketGroups / ixStartPacketGroups / ixStopPortPacketGroups / ixStopPacketGroups*
- *Statistics*
 - *ixCollectStats*

Setup

The data capture and statistics setup commands should be performed before any data capture operations are started.

ixSetPortCaptureMode / ixSetCaptureMode

These commands sends a message to the IxServer to set the receive mode of a single port or list of ports to Capture mode. This mode must be used when traffic is to be captured in the capture buffer. This mode is mutually exclusive with the Packet Group receive mode. The format of these commands are as follows:

ixSetPortCaptureMode *chassisID cardID portID [write]*

ixSetCaptureMode *portList [write]*

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports. The *write* argument commits the settings to the hardware immediately.

Refer to “*ixSetPortCaptureMode*” on page C-113 and “*ixSetCaptureMode*” on page C-101 for a full description of these commands.

ixSetPortPacketGroupMode / ixSetPacketGroupMode

These commands send a message to the IxServer to set the receive mode of a single port or list of ports to Packet Group mode. This mode must be used when real-time latency metrics are to be obtained. This mode is mutually exclusive with the Capture receive mode, for some modules. The format of these commands are as follows:

```
ixSetPortPacketGroupMode chassisID cardID portID [write]  
ixSetPacketGroupMode portList [write]
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports. The *write* argument commits the settings to the hardware immediately.

Refer to “*ixSetPortPacketGroupMode*” on page C-119 and “*ixSetPacketGroupMode*” on page C-107 for a full description of these commands.

ixSetPortDataIntegrityMode / ixSetDataIntegrityMode

These commands send a message to the IxServer to set the receive mode of a single port or list of ports to Data Integrity mode. The format of these commands are:

```
ixSetPortDataIntegrityMode chassisID cardID portID [write]  
ixSetDataIntegrityMode portList [write]
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports. The *write* argument commits the settings to the hardware immediately.

Refer to “*ixSetPortDataIntegrityMode*” on page C-115 and “*ixSetDataIntegrityMode*” on page C-103 for a full description of these commands.

ixSetPortSequenceCheckingMode / ixSetSequenceCheckingMode

These commands send a message to the IxServer to set the receive mode of a single port or list of ports to Sequence Checking mode. The format of these commands are as follows:

```
ixSetPortSequenceCheckingMode chassisID cardID portID [write]  
ixSetSequenceCheckingMode portList [write]
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports. The *write* argument commits the settings to the hardware immediately.

Refer to “*ixSetPortSequenceCheckingMode*” on page C-123 and “*ixSetSequenceCheckingMode*” on page C-128 for a full description of these commands.

ixClearTimeStamp

The **ixClearTimeStamp** command sends a message to the IxServer to synchronize the timestamp on a group of chassis. This feature is useful for calculating latency on ports across chassis. The format of this command is as follows:

ixClearTimeStamp *portList*

where *portList* identifies a number of ports.

Refer to “*ixClearTimeStamp*” on page C-39 for a full description of this command.

ixClearPortStats / ixClearStats

These commands clear all of the statistics counters on a single port or list of ports (except for the stats in the Latency/Sequence view). The format of these commands is as follows:

ixClearPortStats *chassisID cardID portID*

ixClearStats *portList*

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “*ixClearPortStats*” on page C-34 and “*ixClearStats*” on page C-37 for a full description of these commands.

ixClearPortPacketGroups/ ixClearPacketGroups

These commands clear all of the packet group counters on a single port or list of ports. The format of these commands is as follows:

ixClearPortPacketGroups *chassisID cardID portID*

ixClearPacketGroups *portList*

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “*ixClearPacketGroups*” on page C-26 and “*ixClearPortPacketGroups*” on page C-32 for a full description of these commands.

ixResetSequenceIndex/ ixResetPortSequenceIndex

These commands send a message to the IxServer to reset the sequence index of a single port or a list of ports. The format of these commands are as follows:

ixResetPortSequenceIndex *chassisID cardID portID [write]*

ixResetSequenceIndex *portList [write]*

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports. The *write* argument commits the settings to the hardware immediately.

Refer to “*ixResetSequenceIndex*” on page C-92 and “*ixResetPortSequenceIndex*” on page C-90 for a full description of these commands.

Capture Data

ixStartPortCapture / ixStartCapture / ixStopPortCapture / ixStopCapture

These commands start and stop port capture on a single port or on a group of ports. The format of these commands is as follows:

```
ixStartPortCapture chassisID cardID portID
ixStartCapture portList
ixStopPortCapture chassisID cardID portID
ixStopCapture portList
```

where *chassisID*, *cardID*, and *portID* identify a single port and *portList* identifies a number of ports.

Refer to “*ixStartPortCapture*” on page C-147, “*ixStartCapture*” on page C-138, “*ixStopPortCapture*” on page C-170 and “*ixStopCapture*” on page C-161 for complete descriptions of these commands.

ixStartPortPacketGroups / ixStartPacketGroups / ixStopPortPacketGroups / ixStopPacketGroups

These commands start and stop calculation of real-time latency metrics on a single port or on a group of ports. Both packet groups and wide packet groups count the number of frames received per packet group ID (PGID) and calculate the minimum, maximum and average latencies. The format of these commands is as follows:

```
ixStartPortPacketGroups chassisID cardID portID
ixStarts portList
ixStopPortPacketGroups chassisID cardID portID
ixStopPacketGroups portList
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “*ixStartPacketGroups*” on page C-143, “*ixStartPacketGroups*” on page C-143, “*ixStopPortPacketGroups*” on page C-174 and “*ixStopPacketGroups*” on page C-166 for complete descriptions of these commands.

Statistics

ixCollectStats

This command gathers the same specified statistic from a number of ports and places the results in a return array. The format of this command is as follows:

ixCollectStats *portList statName RxStats TotalStats*

where *portList* identifies a number of ports to collect statistics from, *statName* is the name of the statistic to collect, *RxStats* is the returned array of statistics and *TotalStats* is the returned total number of frames (that is, the sum of *RxStats*). *statName* must match one of the standard options defined in the *stat* command (see “*stat*” on page A-703). *RxStats* is an array whose indices are the ports over which the statistics were collected.

Note that the *RxStats* indices are separated by commas and not spaces as in other array references used with maps. Also recall that most of the statistics collected are 64-bit values, as indicated in the *stat* command. Calculations on these values should be performed using the *mpexpr* command.

Refer to “*ixCollectStats*” on page C-41 for a full description of this command.

ixRequestStats

This command requests that the statistics associated with a list of ports or a port map be retrieved at the same time. The statistics are then read using the *statList* command. The format of the command is as follows:

ixRequestStats *portList*

where *portList* identifies a map name or list of ports.

ARP

All of the commands in this section require that the *ip* command be used on the port(s) before any ARP command.

ixEnableArpResponse / ixEnablePortArpResponse

These commands enable ARP response to ARP requests on a single port or list of ports. The format of these commands is as follows:

```
ixEnablePortArpResponse chassisID cardID portID  
ixEnableArpResponse portList
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “*ixEnableArpResponse*” on page C-58 and “*ixEnablePortArpResponse*” on page C-60 for a full description of these commands.

ixDisableArpResponse / ixDisablePortArpResponse

These commands disable ARP response to ARP requests on a single port or list of ports. The format of these commands is as follows:

```
ixDisablePortArpResponse chassisID cardID portID  
ixDisableArpResponse portList
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “*ixDisableArpResponse*” on page C-52 and “*ixDisablePortArpResponse*” on page C-54 for a full description of these commands.

ixClearPortArpTable / ixClearArpTable

These commands clear all of the statistics counters on a single port or list of ports. The format of these commands is as follows:

```
ixClearPortArpTable chassisID cardID portID  
ixClearArpTable portList
```

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports.

Refer to “*ixClearStats*” on page C-37 and “*ixClearPortStats*” on page C-34 for a full description of these commands.

ixTransmitPortArpRequest / ixTransmitArpRequest

These commands signal the protocol server to start transmission of ARP requests as indicated through the *arpServer* command on a single port or list of ports. The format of these commands is as follows:

ixTransmitPortArpRequest *chassisID cardID portID*

ixTransmitArpRequest *portList*

where *chassisID*, *cardID*, and *portID* identifies a single port and *portList* identifies a number of ports. Refer to “*ixTransmitPortArpRequest*” on page C-185 and “*ixTransmitArpRequest*” on page C-183 for a full description of these commands.

Console Output and Logging

The commands in this section relate to textual output to the console and to the operation of the log file. The commands covered in this section are as follows:

- *Console Output*
 - *ixPuts*
- *Logging*
 - *logOn / logOff*
 - *logMsg*
 - *enableEvents*

Error messages

ixErrorInfo

Refer to “*ixErrorInfo*” on page C-64 for a full description of this global variable. The `$::ixErrorInfo` global variable holds the text associated with the error return from most TCL API commands. For example:

```
ixPuts $::ixErrorInfo
```

errorMsg

The `errorMsg` command outputs its arguments to the error file with or without a newline. The format of the command is as follows:

```
errorMsg [-nonewline] arg...
```

where `-nonewline` indicates that a newline should not be appended to the output and `arg...` is any number of arguments, which is concatenated and logged.

Refer to “*errorMsg*” on page B-14 for a full description of this command.

Console Output

ixPuts

Refer to “*ixPuts*” on page C-88 for a full description of this command. The `ixPuts` command outputs its arguments to the console window with or without a newline. The format of the command is as follows:

```
ixPuts [-nonewline] arg...
```

where `-nonewline` indicates that a newline should not be appended to the output and `arg...` is any number of arguments, which is concatenated and printed.

Logging

logOn / logOff

These commands enable and disable logging. The `logOn` command also defines the name of the log file. The format of these commands are as follows:

logOn *filename*

logOff

where *filename* is the name of the log file to be created.

Refer to “*logOn*” on page B-21 and “*logOff*” on page B-20 for a full description of these commands.

logMsg

The **logMsg** command outputs its arguments to the log file with or without a newline. The format of the command is as follows:

logMsg [*-nonewline*] *arg...*

where *-nonewline* indicates that a newline should not be appended to the output and *arg...* is any number of arguments, which is concatenated and logged.

Refer to “*logMsg*” on page B-19 for a full description of this command.

enableEvents

This command enables or disables the creation of a separate log file to hold errors and warnings produced by API calls. The log file created includes the time and date of creation and is held in *C:\Program Files\Ixia*. This feature is enabled by default on Windows-based machines and disabled by default on Unix-based machines.

Refer to “*enableEvents*” on page B-13 for a full description of this command.

Port CPU Control

The API commands related to controlling code and command execution on port CPUs is documented in [Port CPU Control](#) on page 5-123. This section discusses a high-level API command which may be used as a replacement for the [*pcpuCommandService*](#).

Issue Port CPU Commands

issuePcpuCommand

The *issuePcpuCommand* command executes a Linux command on a set of ports. Refer to [*issuePcpuCommand*](#) on page C-5 for a complete description of this command. The format of this command is as follows:

issuePcpuCommand *portList command*

where *portList* is a TCL list of ports passed in by reference and *command* is the text of the command to be executed, which must use an absolute path. For example, ‘/bin/ls’. No filename expansion is performed on the command. For example, ‘/bin/ls /bin/ix*’ finds no matches. This, and the restriction on absolute path, may be avoided by executing the command through a *bash* shell, as in the following example:

```
set portList [list [list 1 1 1] [list 1 1 2]]  
issuePcpuCommand portList "/bin/bash -c 'ls -l /bin/ix*'"
```

The result of the command’s execution indicates whether the command was sent to the ports or not. No indication is given that the ports actually ran successfully on the ports. The individual port by port result of the command can be retrieved by using the *getFirst* / *getNext* functions of [*pcpuCommandService*](#).

Miscellaneous Commands

Several additional commands are available. The commands in this category are described in the following table:

Table 3-4. Miscellaneous Commands

Command	Arguments	Usage
<i>ixIsOverlappingIpAddress</i>	ipAddress1 count1 ipAddress2 count2	Determine whether two IP address ranges overlap.
<i>ixIsSameSubnet</i>	ipAddress1 mask1 ipAddress2 mask2	Determine whether two IP subnets overlap.
<i>ixIsValidHost</i>	ipAddress mask	Determines whether the host part of a masked address is valid.
<i>ixIsValidNetMask</i>	mask	Determines a net mask validity.
<i>ixIsValidUnicastIp</i>	ipAddress	Determines Unicast IP address validity.
<i>ixConvertFromSeconds</i>	time hours minutes seconds	Convert <i>time</i> into <i>hours</i> , <i>minutes</i> and <i>seconds</i> .
<i>ixConvertToSeconds</i>	hours minutes seconds	Converts a number of hours, minutes and seconds into a number of seconds.

4

Programming

API Structure and Conventions

This chapter discusses general structure of the Ixia Tcl commands and suggested programming sequence.

Most of the Tcl commands have the following same basic structure:

- A number of configuration options that are used to set test and other parameters.
- A standard set of options that push the data options toward the hardware and read information back from the hardware.
- Additional command specific options used to perform special settings or operations.

Standard Sub-Commands

The standard sub-commands that come with most commands are mentioned in the following table:

TABLE 4-1. Standard Options

Method	Usage
config	Sets a specified value to a specific option, which is most often a desired hardware setting. The value is stored in an object in IxTclHAL temporarily.
cget	Gets a specified option's value, which was stored in the IxTclHAL object.
set	Information is transferred from the IxTclHAL object to the IxHal software layer, but not sent to the hardware. The <code>set</code> method takes as arguments the chassis ID, card number and port number being addressed.

TABLE 4-1. Standard Options

Method	Usage
write	Information previously transferred to the IxHal software layer is sent to the hardware. The <code>write</code> method takes as arguments the chassis ID, card number and port number being addressed. Although each class provides its own <code>write</code> method, it is usually more convenient to call <code>ixWriteConfigToHardware</code> , which sends all outstanding set's to the hardware at the same time.
get	Information from the hardware is read out to the IxHal layer and into the member variables. In many cases, the IxHal layer holds more information than is represented in a single set of member variables and additional methods are needed to obtain more data. The <code>get</code> method takes as arguments the chassis ID, card number and port number being addressed.
setDefault	Default values for the members are set.
decode	A captured frame is analyzed and appropriate member variables are set to reflect the contents of the frame.

In general, hardware parameters may be saved through the use of a ‘config’ and ‘set’ option and then retrieved at any later time by a ‘get’ option followed by a ‘cget’ option. This is because the IxHal level maintains memory of all of the settings. This relationship of methods is illustrated in *Table 4-1 on page 4-1*.

Note that a single instance of each command exists, with a set of associated data variables, called *standard options*. The standard options from one command are often used in another. For example, the `ipAddressTable` command uses the standard options from the `ipAddressTableItem` command. The most recent standard options from the `ipAddressTableItem` command are used by the `ipAddressTable` command. Ensure that the standard options from dependent commands are set immediately before being used. Intervening commands may interfere.

The values defined in the tables for each of the API commands may be used in the following ways:

- As an argument to a *config* command. For example:

```
port config -masterSlave portMaster -and-
port config -masterSlave $::portMaster
```

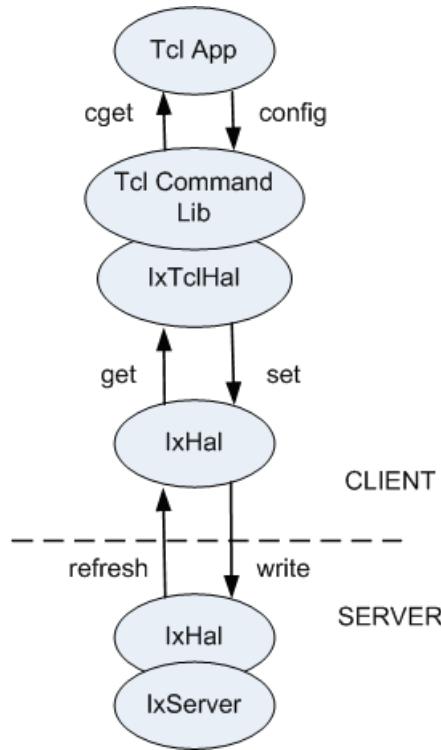
are valid. In the first case, the `port config` command figures out the value of `portMaster` (0). In the second case, the global variable `$::portMaster` (which is defined in the *IxTclHal* package) is used to determine a value of 0. The `::` qualifier indicates that the variable is defined in the global context.

- As a variable used for comparison. For example:

```
port get 1 1 1
set msValue [port cget -masterSlave]
if [$msValue == $::portMaster] ...
```

Here the \$:: form must be used to refer to the value of *portMaster*.

Figure 4-2. Standard Method Relationships



Standard Return Codes

All commands in the TCL API use a common set of return codes. These codes are listed in [Table 4-2](#). These codes are global TCL variables, which may be referred with a preceding '\$' (for example, \$ixTcl_ok) in a global context or a preceding '\$::' (for example, \$::ixTcl_ok) in any context. The symbolic codes should be used in preference to literal values.

TABLE 4-2. TCL API Return Codes

Code	Value	Usage
ixTcl_ok TCL_OK	0	No error, successful return.
ixTcl_generalError TCL_ERROR	1	An error has occurred.
ixTcl_versionMismatch	2	The software version for the TCL API does not match that used on a connected chassis.
ixTcl_chassisTimeout	3	A timeout occurred while connecting to a chassis.
ixTcl_notAvailable	100	A port may not exist or may be in use by another user.
ixTcl_unsupportedFeature	101	The port does not support a feature.

TABLE 4-2. TCL API Return Codes

Code	Value	Usage
ixTcl_outOfMemory	102	The TCL execution has run out of main memory.
ixTcl_addedAsDisabled	103	The entry was added, but was disabled due to quantity or volume constraints.
ixTcl_notLicensed	104	Not licensed
ixTcl_noWriteRequired	200	Writing not required
ixTcl_invalidChassisChain	201	Invalid chassis chain
ixTcl_hardwareConflict	202	When adding a load module, a duplicate load module serial number was encountered.

Sequence of Steps

The following sequence of steps should be followed to write a successful Tcl script:

1. Load the IxTclHal package.

The IxTclHal package contains all the Ixia Tcl Library commands. After loading this package, these commands are made available in the test script. The format of using the package command is as follows:

```
package require IxTclHal
```

Note that this adds a large number of commands to Tcl, which causes the ‘?’ command (list available commands) to take up to a minute to execute.

2. Connect to the chassis on which the test is to be executed.

After loading the package, the chassis has to be connected to where the test is going to be executed. The following commands are used to connect and set up the chassis parameters:

```
chassis add <hostname or IP address>
chassis config -id <chassisID>
chassis set <hostname or IP address>
```

The *chassis add* command connects to the chassis. The *chassis config -id* command associates a numeric ID with the chassis. The *chassis set* command sets the ID of the chassis in IxHAL. It is important to assign a chassisID to the chassis as it is used in the *map* command. If multiple chassis are to be used, then multiple *chassis add* commands must be given and each chassis should be assigned a unique ID.

Alternatively, the following sequence could be used:

```
ixConnectToChassis <hostname or IP address>
set chassisID [ixGetChassisID <hostname or IP address>]
```

The *ixConnectToChassis* takes care of all three steps in the previous example, assigning a chassis ID on its own. The call to *ixGetChassisID* is needed to retrieve the assigned chassis ID for future use.

3. Set up the traffic mapping

This is an optional step. The mechanism for setting up traffic mapping is provided only for convenience. You may use your own methods for storing this information.

Before any test can be executed, it is important to specify the flow of traffic, that is, the transmit and receive ports. The mapping for these ports is specified using the *map* command as follows:

```
map config -type oneZone; # or one2many, many2one,
many2many
map add <TxChassisID> <TxCardID> <TxPortID> <RxChassisID>
<RxCardID> <RxPortID>
```

This command stores the transmit chassis, card, port and receive chassis, card, port combinations in a Tcl array within the scope of the Tcl script. There are four types of mappings, as mentioned in the following list:

a: One to One mapping

- b:** One to Many mapping
- c:** Many to One mapping
- d:** Many to Many mapping

For the mappings specified in a), b), c), and d) above, the chassis, card, port combinations are stored in Tcl arrays *one2oneArray*, *one2manyArray*, *many2oneArray* and *many2manyArray*, respectively.

Each Transmit/Receive combination in *one2oneArray* is unique. That is, there is only one Receive port for each Transmit port. The Receive port may also be set as Transmit port. Similarly, for the *one2manyArray*, any Transmit port cannot be used as a Receive port for a different set, and for the *many2oneArray*, any Receive port cannot be used in a different set of the many-to-one map. The *many2manyArray* can contain any combination of transmit and receive ports. A port can be assigned to be a Receive port for any number of Transmit ports and can also act as a Transmit port for several Receive ports.

TABLE 4-3. Traffic Map Array

Array Type	Format
one2oneArray	one2oneArray(txChassis,txCard,txPort) {{rxChassis rxCard rxPort}}
one2manyArray	one2manyArray(txChassis,txCard,txPort) {{rxChassis1 rxCard1 rxPort1} {rxChassis2 rxCard2 rxPort2} {rxChassisN rxCardN rxPortN}}
many2oneArray	many2oneArray(rxChassis,rxCard,rxPort) {{txChassis1 txCard1 txPort1} {txChassis2 txCard2 txPort2} {txChassisN txCardN txPortN}}
many2manyArray	many2manyArray(txChassis,txCard,txPort) {{rxChassis1 rxCard1 rxPort1} {rxChassis2 rxCard2 rxPort2} {rxChassisN rxCardN rxPortN}}

The *map* command is very useful when writing scripts. Upon closer inspection, it is apparent that the Transmit ports in the traffic flow are all stored as elements of the arrays (except for *many2oneArray*) and the Receive ports are stored as values (Tcl Lists) of these arrays. This method of storage allows a great deal of flexibility when this information is needed. In Tcl, the command [*array names one2oneArray*], for example, gives access to all the Transmit ports and to access the Receive port, *\$one2oneArray(\$txChassis,\$txCard,\$txPort)* gives the list with the Receive chassis, card, port combination.

4. Set up test related parameters

The test related information such as duration of test, number of trials, configuration of learn frames and IP/IPX addresses may be set up next.

5. Configure the port parameters

The port parameters such as speed, duplex, loopback and auto-negotiation must be set in IxHAL and then in hardware (by sending the message to IxServer). The following steps should be followed to configure the ports:

```
port config -autonegotiate      true
port config -duplex            full
port config -numAddresses      1
port config -MacAddress        {00 01 02 03 04 05}
port set $chassisID $cardID $portID
port write $chassisID $cardID $portID
```

The addresses for the ports are assigned in this step. Note that for the Tcl scripts, the addresses are assigned to ports but they are actually configured in the streams (see step 6). This is due to the fact that when executing tests that send and receive traffic from switches and routers, addresses are assigned to physical ports. The concept of streams is invisible to the switches and routers. The MAC address is assigned to the port using the *port config -MacAddress* command. The source and destination IP addresses are assigned to ports using the *ip* command. Similarly, the IPX network and node addresses and sockets are assigned to ports using the *ipx* command. The following example shows the assignment of IP addresses to a port:

```
ip config -sourceIpAddr    198.18.1.100
ip config -destDutIpAddr   198.18.1.1
ip config -destClass       classC
ip config -sourceClass     classC
ip set $chassisID $cardID $portID
```

6. Configure the streams on the transmit ports

The traffic is sent in streams, which contain the frame characteristics. Multiple streams may be created per port. The important parameters in the stream are frame size, inter-frame gap, frame data type, the number of frames to be transmitted, the source and destination MAC addresses in each frame and whether they are incrementing, decrementing or fixed. When configuring the protocol related parameters such as MAC, IP or IPX addresses, the protocol configuration is not be written to hardware until a *stream set* command is used. In addition, the User Defined Fields (UDFs) can be configured to overlay a 1 to 4 byte custom pattern over the specified frame data. Examples of usage for UDFs include setting up filters on the receive ports for a particular UDF pattern, allowing an incrementing IP address or IPX socket, and adding a sequence ID to the frame.

The following code shows stream configurations:

```
stream config -numFrames 10000
stream config -name "MyStream"
stream config -dma stopStream

# Calculate the inter-frame gap using the utility command,
calculateGap
stream config -ifg [calculateGap $rate $framesize
$preambleSize $speed]

# get the transmit chassis, card, port combination from the
Tcl
# array created by the map command by using [array names
<mapArray>].
# For example, the txChassis,txCard,txPort combination for
the first
# set in the one2oneArray can be obtained as follows:
```

```

# set txMap [lindex [array names one2oneArray] 0]
# scan $txMap "%d %d %d" txChassis txCard txPort
port get $txChassis $txCard $txPort
set txPortMacAddress [port cget -MacAddress]

# The source MAC address of the port is set in the stream
stream config -sa $txPortMacAddress

# The destination MAC address of this transmit port can be
obtained
# from the receive port by using:
# set rxMap $<mapArray> ($txChassis $txCard $txPort)
# scan rxMap "%d %d %d" rxChassis rxCard rxPort
port get $rxChassis $rxCard $rxPort
stream config -da [port cget -MacAddress]

# overwrite 4 bytes of the frame data at offset 42 with
magic
# pattern "BE EF" using UDF 4
udf config -offset 42
udf config -enable true
udf config -countertype c8x8
udf config -initval "BE EF"
udf set 4

# set the current stream configuration in IxHAL as the
first stream
# on this port
stream set $txChassis $txCard $txPort 1

```

7. Configure the filters parameters on receive ports

The filters are used to count or capture desired format of frames. To capture the frames, the capture trigger and capture filter parameters have to be enabled. Two counters, User Defined Statistics Counter 1 and User Defined Statistics Counter 2, can be enabled to count frames that match the defined constraints. The UDF values that are set using the stream command can be filtered upon and counted using these counters. To define the constraints on these counters, the filterPallette command can be used to specify up to two Destination MAC addresses, two Source MAC addresses and up to two patterns.

```

# Enable the User Defined Statistics Counter 1, Capture
# trigger and capture filter counters
filter config -userDefinedStat1Enable true
filter config -captureFilterEnable true
filter config -captureTriggerEnable true

# set the User Defined Statistics Counter 1 to count frames
# that have destination MAC address "00 01 02 03 04 05"
filter config -userDefinedStat1DA "00 01 02 03 04 05"

# set the capture filter counter to capture frames
# that have pattern "BE EF"
filter config - captureFilterPattern "BE EF"

# set up the filter pallette
filterPallette config -DA1 "00 01 02 03 04 05"
filterPallette config -pattern1 "BE EF"
filterPallette config -patternOffset1 42

```

```

# obtain the receive chassis, card, port combination and
set the
# filter and filter pallette on the receive port
filter set $rxChassis $rxCard $rxPort
filterPallette set $rxChassis $rxCard $rxPort

```

8. Write the configuration into hardware

After the stream and filter configurations are set in IxHAL, a message must be sent to IxServer to commit these configurations to hardware. Every command has a write sub-command that writes that command related information into the hardware. However, it is inefficient to commit to hardware after updating every parameter as it might effect the performance of system. A more efficient method of writing to hardware is to update all the IxHAL objects first and send out one message to IxServer. For example, after all the chassis, ports, filters and stream information is set up in IxHAL, a message can be sent to the IxServer requesting it to write every configuration on a chassis.

We recommend one of the following two methods of writing to hardware:

- a:** If a traffic map was set-up in step (3):

```
ixWriteConfigToHardware <map>
```

Here the *<map>* is one of *one2oneArray*, *one2manyArray*, *many2oneArray* or *many2manyArray*.

- b:** If a traffic map was not set-up in step (3):

```
set portlist {{1 1 1} {1 1 2} {1 1 3} {1 1 4}}
ixWriteConfigToHardware portlist
```

The *portlist* variable is a list of lists, each member containing three elements: chassis, card, and port.

Note that *ixWriteConfigToHardware* does not send port configuration changes (for example, mii settings, port speed or auto-negotiation) to the hardware. If changes have been made to these parameters, use *ixWritePortsToHardware*. However, this may result in a loss of link, depending on the changes that have been made.

9. Start the transmission

Now that the configuration is set in hardware, transmission of the streams on all the involved ports is started. A utility command *startTx* is provided that starts the transmission on all the ports simultaneously. *StartTx* command is called with the array created by the map command (*one2oneArray*, *one2manyArray*, *many2oneArray*, or *many2manyArray*) as an argument. This command uses the *portGroup* command to add all the Transmit ports in the array to a group with a given ID. This group ID is then used to send a message to the IxServer to start transmission at the same time.

10. Validate the received frames on receive ports

Usually, only frames that were transmitted must be counted to obtain reasonable results. The DUT may be transmitting management frames periodically, which should not be counted in the validating scheme of the test. As discussed earlier, the UDFs and filters are used in streams to achieve this. The statistics counters such as the User Defined Statistics Counters are used

to count the valid frames and calculations may be performed to get desired results. Frames may also be captured that may be decoded and counted.

11. Output results

Finally, the results may be obtained from the statistics or frames captured in the capture buffer and stored in any desired format.

How to write efficient scripts

A script is a logical sequence of operations that are sequentially executed. However, a script is an application program that must be designed carefully just like a normal C or C++ program. Because a Tcl script does not need to be compiled as in a C/C++ code, it is actually more difficult to get the best optimized code for a Tcl script.

Also, compared to a GUI application where events take place very slowly (clicking on buttons using a mouse), a script passes through the same events very quickly. This speed of execution may cause time synchronization problems with the IxServer application on the chassis. Therefore, it is very important to implement the sequence of events in a very intelligent way so as to achieve the most optimized execution of these events.

Before writing any script, it helps to first design the logic of the test using flow-charts or similar methods. More importantly, the logic should first be implemented in the IxExplorer GUI application to test the algorithm of the script. Therefore, it is of paramount importance that you understand the concepts of streams, filters, UDFs, capture buffers, and so on before writing any script. These are described in the *Ixia Reference Guide*.

It is important to follow the sequence of steps outlined in “*Sequence of Steps*” on page 4-5. The following list provides the reasons:

- 1.** It is important to configure the Port Properties in the very beginning of the script. For example, on a 10/100 card, there is an autonegotiation parameter. If this parameter is to be turned on, then during the auto-negotiation process, the link goes down and it takes a small amount of time to come up again. If there are a number of ports involved, the setting of port properties can take a long time. Therefore, we recommend that the ports be set first. This is done using the *port set* and *port write* commands. Do not implement this step in the middle of the script since it only slows down the total time of execution of the test.
- 2.** To send and receive traffic on the ports, the transmit and receive port mappings must be defined and the port numbers stored in a structured array or list. This way, when streams and filters are to be configured, this array or list only needs to be looped through.
- 3.** Before creating streams on the transmit ports, decide how many streams are needed to achieve the traffic profile to be used. It is possible that only one stream is needed in most cases. If not certain, use the IxExplorer GUI application to create the stream and transmit frames on it. Try to avoid creating streams for every iteration of the test. Sometimes it is easier to create all the necessary streams and simply disable the unneeded ones during the iterations.
- 4.** To verify the filters, use the IxExplorer GUI to first mimic the situation. Use magic numbers in the payload of the frames wherever possible so that only the frames involved in the test are received. The magic numbers eliminate the

inclusion of loopback, management frames or Spanning Tree BPDUs, for example, forwarded by the DUT.

5. If your script is running in a small loop and requesting data very quickly from IxServer, it is possible IxServer is not be able to keep up with the requests. If your script must request statistics or other data in a small `for` or `while` loop, consider using delays using the Tcl `after` command.

Multi-Client Usage

It is occasionally helpful to run IxExplorer at the same time as the Tcl Development environment. IxExplorer can provide instant visual verification. When doing so, it is important to perform a Chassis Refresh operation in IxExplorer after executing the following Tcl setup commands:

- `ixSetCaptureMode`
- `ixSetPacketFlowMode`
- `ixSetPacketGroupMode`
- `ixSetPacketStreamMode`
- `ixSetPortCaptureMode`
- `ixSetPortPacketFlowMode`
- `ixSetPortPacketGroupMode`
- `ixSetPortPacketStreamMode`
- `ixWriteConfigToHardware`
- `ixWritePortsToHardware`

Mpexpr versus Expr

Statistics and other values used in the Ixia Tcl environment are 64-bit as opposed to 32-bit values. It is important to use `mpexpr`, as opposed to `expr`, to calculate expressions and maintain 64-bit accuracy. The 64-bit values are indicated in the individual descriptive pages for the following commands:

- `captureBuffer`
- `packetGroupStats`
- `portGroup`
- `stat`

5

IxTclHal API Description

This chapter presents an organized description of the IxTclHAL API commands based on major topics. The main topics covered are the following:

- *Chassis, Cards and Ports*: Basic overhead to set up the test and the hardware.
- *Data Transmission*: Setting up streams and flows to be applied to ports.
- *Data Capture and Statistics*: Setting up conditions to capture received data and statistics.
- *Interface Table*: Setting up interfaces and IP addresses.
- *Port CPU Control*: Setting up and executing code and commands on port CPUs.

All of the commands are covered within these sections, but only the most significant options and sub-commands are discussed. Not all of the options, nor all of the sub-commands can be assumed to be discussed in this chapter. In particular, if not otherwise noted the `get`, `cget`, `config`, `set`, `setDefault`, `decode` and `write` sub-commands are assumed to exist and to perform standard functions.

Appendix A - IxTclHAL Commands includes complete descriptions of each of the IxHal commands.

Chassis, Cards and Ports

These commands included in this section are related to the setup of tests, before any data is applied. As discussed in the *Ixia Reference Guide*, Ixia equipment is organized as a chain of individual chassis connected by Sync-In/Sync-Out wires. The **chassisChain** command is used to hold information about the chain as a whole. One copy should be instantiated for the lifetime of the program. The **chassis** command is used to define and add chassis to the chain. Each chassis has two very important options: **id**, which is referenced elsewhere in referring to all levels of hardware, and **name**, which is the IP hostname/address used to communicate with the hardware. **chassisChain** sub-command **broadcastTopology** should be called after all the chassis have been added to the chain. Although each individual chassis, card and port has an individual write method, **ixWriteConfigToHardware** is a convenient means of writing to all chassis, in synchronization.

With the advent of the IXIA 100, the means by which geographically distributed chassis chains may be synchronized has been expanded. This is controlled by the **timeServer** command.

Cards reside within chassis and the **card** command is provided to access several read-only version variables for the card.

Ports are the principal focus of setup programming in the TCL API. All of the port's characteristics are visible and changeable through **port** and its associated commands.

The following commands are included in this section:

- *session*: Used to control user login and sharing.
- *version*: Provides version information about the running software.
- *chassisChain*: Controls the handling of the chassis chain that contains one or more chassis.
- *timeServer*: Allows the selection of the timing source for a chassis.
- *chassis*: Handles the chassis that contains multiple cards.
- *card*: Handles a card that contains one or more ports.
- *port*: Controls the basic features of a port. Subsidiary commands are used for special port features.
 - *MII*: This set of commands controls access to the MII registers associated with some ports.
 - *mii*: Controls basic access.
 - *miae*: Controls extended access.
 - *mmd*: Controls access to MMI devices.
 - *mmdRegister*: Controls access to MMD registers.
 - *xauI*: 10GE XUAI configuration.

- *Packet over Sonet*: This set of commands controls SONET related parameters.
 - *sonet*: Controls basic sonet parameters.
 - *sonetError*: Allows errors to be inserted in SONET streams.
 - *sonetOverhead*: Controls SONET overhead parameters.
 - *dcc*: Controls placement of DCC bytes in the SONET overhead.
 - *RPR*: Controls SRP encapsulation and SRP specific control messages.
 - *ppp and pppStatus*: Controls and monitors point to point protocol operation.
 - *hdlc*: Controls HDLC header formatting.
 - *frameRelay*: Controls frame relay header formatting.
 - *bert and bertErrorGeneration*: Controls bit error rate testing (BERT) and error generation.
 - *bertUnframed*: Controls unframed bit error rate testing.
- *ATM*: This set of commands controls ATM specific parameters.
 - *atmPort*: Controls port general parameters
 - *atmHeader*: Controls ATM header parameters.
 - *atmHeaderCounter*: Controls variations of the VPI and VCI values in an ATM header.
- *10GE*
 - *Link Fault Signaling*: This set of commands controls link fault signal insertion.
 - *linkFaultSignaling*: Controls the insertion process.
 - *customOrderedSet*: Defines custom signal messages.
 - *txRxPreamble*: Controls the preamble transmit and receive settings.
 - *Optical Digital Wrapper / FEC*: Enables use of the optical digital wrapper and FEC errors.
 - *opticalDigitalWrapper*: Enables the wrapper.
 - *fecError*: Inserts errors for FEC error detection.
 - *CDL Support*: Use of Cisco Converged Data Layer (CDL)
 - *cdlPreamble*: Controls the contents of the CDL preamble.
 - *xfp*: XFP settings associated with UNIPHY-XFP ports.
 - *lasi*: LASI settings associated with XENPAK ports.
- *portGroup*: A collection of ports, which allows simultaneous action across the set of ports.

session

session is an optional command used to control sharing of ports on one or more chassis. It should be used where there is any possibility of multiple users sharing chassis. **session -login** is used to log-in and **portGroup -setCommand** is used to take ownership of ports. See [session](#) on page A-653 for full details.

The important options and sub-commands of this command are listed in the table below.

TABLE 5-1. session Options

Member	Usage
userName	The user's name after login.
captureBuffer SegmentSize	Sets the capture buffer request size in MB.

TABLE 5-2. session Sub-Commands

Member	Usage
login	Logs a user in for purposes of ownership.
logout	Logs out the current user.

version

version provides access to assorted pieces of version information for the Tcl software. Note that on Unix systems, a connection to the chassis must have occurred before version information is available. See [version](#) on page A-838 for full details and [ixConnectToChassis](#) on page C-44 for connection information.

chassisChain

A single instance of this command should be instantiated and not destroyed for the entirety of the test process. It is the container that holds all of the individual chassis designations and their connections. See the *Ixia Reference Guide* for a discussion of chassis chains. See [chassisChain](#) on page A-107 for full details.

The important options and sub-commands of this command are listed in the table below.

TABLE 5-3. chassisChain Options

Member	Usage
startTime	The delay time before port transmit starts.

TABLE 5-4. chassisChain Sub-Commands

Member	Usage
broadcastTopology	Must be called after the last chassis has been added with <code>chassis.add</code> .

timeServer

The **timeServer** command handles the means by which chassis chains are coordinated. See the *Ixia Reference Guide* for a discussion of timing sources. Refer to [timeServer](#) on page A-811 for details. A chassis chain may use any of the following time sources:

- *Internal*: Internally generated by the chassis.

- *GPS Server*: Generated by the GPS within an IXIA 100 chassis.
- *SNTP Server*: Generated by a network available SNTP (Simple Network Time Protocol) server.
- *CDMA Server*: Generated by the CDMA unit within an IXIA 100 chassis.

The important options and sub-commands of this class are listed in the table below.

TABLE 5-5. timeServer Command Options

Member	Usage
timeSource	The choice of time source.
sntpClient	For the SNTP choice, the location of the SNTP server.
antennaStatus	For the GPS unit, the antenna's connection status.
gpsStatus	For the GPS unit, the locked/unlocked status of the GPS.
gpsTime	For the GPS unit, the GPS read time, in seconds.
pllStatus	For the GPS unit, the status of the phased locked loop that is driven by the GPS.
qualityStatus	For the GPS unit, the quality of the received GPS signal.
state	For the GPS unit, the current state of the GPS.

chassis

chassis is used in the definition of a chassis and addition of the chassis to the chassis chain. See the *Ixia Reference Guide* for a discussion of chassis. See [chassis](#) on page A-99 for full details.

The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-6. chassis Options

Member	Usage
id	The identification number given to the chassis. This is used in most commands to associate with ports.
name	This is the IP hostname or IP address of the chassis, which is used to actually communicate with the chassis. Use 'localhost' if you are running your TCL application on the chassis itself.
sequence	The sequence of a chassis in a chain.

TABLE 5-7. chassis Sub-Commands

Member	Usage
add	Adds a new chassis to the chain.

TABLE 5-7. chassis Sub-Commands

Member	Usage
export	Writes a data file with all card and port configurations to a file which may be used with the <i>import</i> command.
import	Reads and installs a previously written file from the <i>export</i> sub-command.

card

The *card* command retrieves several card characteristics. See the *Ixia Reference Guide* for a discussion of load modules. Refer to *card* on page A-79 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-8. card Options

Member	Usage
fpgaVersion	The FPGA version on the card.
hwVersion	The card's hardware version.
portCount	The number of ports on the card.
type	The type of the card.
typeName	The name of the type of card.
serialNumber	The serial number of the card.
appSId	The application ID.

TABLE 5-9. card Sub-Commands

Member	Usage
write	Card specific properties are written to the card, without any stream or port properties.

port

The *port* command controls the basic aspects of port setup. Some port and protocol specific attributes are included in this command, while other aspects are covered by additional commands in this section. See the *Ixia Hardware & Reference Guide* for a discussion of port hardware characteristics.

Specifically, the following port types have the indicated additional commands that may be used to control additional port features:

- 10/100 and 10GE XAUI/XGMII Mii: *mii*, *miae*, *mmd*, and *mmdRegister*.
- 10GE XauI: *xauI*, *linkFaultSignaling*, *customOrderedSet*, and *txRxPreamble*.
- Packet over Sonet (POS): *sonet*, *sonetError*, *sonetOverhead*, *dcc*, *ppp* and *pppStatus*, *hdlc*, *frameRelay*.
- POS/BERT (Bit Error Rate Testing): *bert* and *bertErrorGeneration*.
- ATM: *atmPort*, *atmHeader*, and *atmHeaderCounter*.

Note that the elements options `DestMacAddress`, `MacAddress` and `numAddresses` are stored as convenience for use by other sub-commands. Do not destroy the port instance until you are completely done with the port. See [port](#) on page A-521 for full details.

The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-10. **port Options**

Category	Member	Usage
Basic	name	The name associated with the port.
	owner	The name of the owner of the port.
	type	(Read-only) The type of the Ixia port. Both speeds and interface types are described.
	loopback	Controls whether the port is in loopback mode or not.
	flowControl	Enables flow control on the port.
	linkState	(Read-only) The current state of the link with the DUT.
	portMode	For ports that support multi-mode operation, the current operational mode.
Transmit	transmitMode	Controls the following basic transmission mode of the port: <ul style="list-style-type: none"> • Packet stream • Packet flow • TCP Round Trip • Advanced Scheduler • Bit Error Rate Testing (BERT) • Mix of SONET DCC and SPE traffic.
	enableRepeatableLastRandomPattern	For ports that support repeatable random feature, this allows streams that used random values to repeat their values again.
	lastRandomSeedValue	
	transmitClockDeviation	For ports that support the frequency offset feature, a transmit frequency deviation.
	preEmphasis	For ports that support pre-emphasis, a percentage pre-emphasis value.

TABLE 5-10. port Options

Category	Member	Usage
Receive	receiveMode	Controls the following basic receive mode of the port: <ul style="list-style-type: none">• Capture• PacketGroup• TCP Round Trips• Data Integrity• First Time Stamp• Sequence Checking• Bit Error Rate Testing (BERT)• SONET DCC• Wide packet group• PRBS packets
Addressing	DestMacAddress	The destination MAC address. Note that <code>port</code> holds this and the next two values as a convenience only for use in other commands. Do not destroy the port instance until you are done using the port.
	MacAddress	The first source MAC address.
	numAddresses	The number of source addresses assigned to the port.
Flows	usePacketFlowImageFile	Controls whether the port is used in stream mode or flow mode. If set to flow mode, then the <code>packetFlowFileName</code> member should be set.
	packetFlowFileName	The name of the file containing the packet flow information.
<i>Pause Control</i>	directedAddress	The address the port listens to for a directed pause message.
	multicastPauseAddress	The address the port listens to for a multicast pause message.
For: 10/100 Ports	autonegotiate	Sets auto-negotiate mode for the port.
	duplex	Controls half / full duplex mode for the port.
	advertise100FullDuplex advertise100HalfDuplex advertise10FullDuplex advertise10HalfDuplex	These four elements control what speeds and duplex are advertised during autonegotiation.
	speed	10 or 100 Mbps.

TABLE 5-10. port Options

Category	Member	Usage
For: Gigabit Ports	rxTxMode	Basic mode for the port are the following: <ul style="list-style-type: none">• Normal• Loopback• Simulate cable disconnect
	aremovedvertise1000Full Duplex	Controls whether gigabit full duplex is advertised during auto negotiation.
	advertiseAbilities	Sets the following type elements advertised during negotiation: <ul style="list-style-type: none">• None• Send only• Send and Receive• Send and/or Receive
	ignoreLink	Causes the port to ignore the link.
	negotiateMasterSlave	Indicates whether master/slave mode should be negotiated.
	masterSlave	If master/slave mode is being negotiated, then this is the indicates the ports desire (master or slave). Otherwise this is the value associated with the link.
	timeoutEnable	Enables autonegotiation timeout.
For: POS Ports	rxCrc	Indicates whether a 16 or 32 bit CRC is to be used on the receive side of the port.
	txCrc	Indicates whether a 16 or 32 bit CRC is to be used on the transmit side of the port.

TABLE 5-11. port Sub-Commands

Member	Usage
getFeature	Determines whether a specific feature is present in the <i>featureList</i> for the port.
isValidFeature	Determines if a port feature is available for the port.
isActiveFeature	Determines whether a port is currently configured correctly to use a feature.
reset	Deletes all streams from a port. Current configuration is not affected. Note: In order for port reset to take effect, stream write or ixWriteConfigToHardware commands should be used to commit the changes to hardware.
setDefault	Sets the port to default values.

TABLE 5-11. port Sub-Commands

Member	Usage
setFactoryDefaults	Sets a consistent set of default values for the port type. The port mode for dual PHY ports is reset to the default.
setModeDefaults	Sets a consistent set of default values for the port type and the current mode of the port. The mode of the port is not affected.
setParam	Operates as in config, but sets a single option.
setPhyMode	For dual PHY ports, which may operate over copper or fiber, this command allows the mode to be selected.

Note: The *setDefault* sub-command sets all options at default values, as indicated in [port](#) on page A-521. These values are a consistent setting for 10/100 ethernet cards and may or may not be appropriate for other cards. In general, the sequence:

```
port setDefault
port set $chassis $card $port
```

fails.

The *setFactoryDefaults* sub-command, which relates to a particular port, sets all options at default values appropriate for the type of port. The sequence:

```
port setFactoryDefaults $chassis $card $port
port set $chassis $card $port
```

always succeed. For multi-type boards, for example, OC192/10GE WAN, the board type is forced to one particular setting and may not be appropriate.

The *setModeDefaults* sub-command, however, leaves the mode of multi-type boards while performing the same operation as *setFactoryDefaults*.

MII

The MII commands are available for 10/100 MII and 10GE XAUI/XGMII ports only. The following commands are included in this set:

- *mii*: Reads and writes values to ‘old-style’ MII PHYs defined in IEEE 802.3. One internal and two external MII PHYs may be managed, mixed with MII AE PHYs.
- *miiae*: Defines, reads and writes to ‘new-style’ MII AE PHYs defined in IEEE 802.3ae. One internal and two external MII AE PHYs may be managed, mixed with MII PHYs. Each MII AE PHY may consist of 32 MMDs (MDIO Manageable Devices), each with up to 64k devices. The MMDs are defined and managed with the *mmd* command and the registers within those devices are managed by the *mmdRegister* command.
- *mmd*: Defines, reads and writes the devices associated with MII AE PHYs.
- *mmdRegister*: Sets the parameters associated with MMD registers.
- *ixMiiConfig utilities*: A set of high level commands used to set several common SerDes functions on 10GE XAUI/XGMII ports.

mii

See [mii](#) on page A-406 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-12. mii Options

Member	Usage
enableManualAutoNegotiate	If set, causes the port to auto-negotiate when the MII registers are written
miiRegister	The MII register number to read/write.
phyAddress	Physical address of the MII register location. -1 for the default.
readWrite	The read/write properties of the register are as following: <ul style="list-style-type: none">• Disabled• Read-Only• Read-Write
registerValue	The value of the selected register.

TABLE 5-13. mii Sub-Commands

Member	Usage
get	This method should be called first, before any <code>cget</code> operations. The register number indicated in <code>miiRegister</code> is read into <code>readWrite</code> and <code>registerValue</code> .
selectRegister	After <code>get</code> is used, this method allows a different register (as indexed by <code>miiRegister</code>) to be made available in <code>readWrite</code> and <code>registerValue</code> .
set	Sets the values from <code>readWrite</code> and <code>registerValue</code> to be written to the MII register indexed by <code>miiRegister</code> .
write	Sends all modified MII registers to the hardware.

miiae

See [miiae](#) on page A-411 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-14. miiae Options

Member	Usage
phyAddress	Physical address of the MII register location.

TABLE 5-15. miiae Sub-Commands

Member	Usage
clearAllDevices	Removes all associated devices from the MII.
addDevice	Adds a device defined in the <code>mmd</code> command to the MII.

TABLE 5-15. miiac Sub-Commands

Member	Usage
delDevice	Removes a single MMD from the MII.
getDevice	Retrieves the information about a single MMD in the MII. The data about the device is available through the use of the <code>mmd</code> and <code>mmdRegister</code> commands.
set	Sets the devices associated with one of the three supported PHYs: Internal, External1, or External2.
get	Gets the devices associated with one of the three supported PHYs: Internal, External1, or External2.

mmd

See [mmd](#) on page A-414 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-16. mmd Options

Member	Usage
address	Address of the MMD device within its associated MII.
name	Arbitrary name of the MMD device.

TABLE 5-17. mmd Sub-Commands

Member	Usage
clearAllRegisters	Removes all associated registers from the MMD device.
addRegister	Adds a register defined in the <code>mmdRegister</code> command to the MMD.
delRegister	Removes a single register from the MMD.
getRegister	Retrieves the information about a single register in the MMD. This must have been preceded by an <code>miiac getRegister</code> command. The data about the device is available through the use of the <code>mmdRegister</code> command.

mmdRegister

See [mmdRegister](#) on page A-415 for full details. The important options of this command are:

TABLE 5-18. mmdRegister Options

Member	Usage
address	Address of the register location.
name	Arbitrary name of the register.

TABLE 5-18. mmrdRegister Options

Member	Usage
readWrite	The read/write properties of the register: <ul style="list-style-type: none">• Disabled• Read-Only• Read-Write
registerValue	The value of the selected register.

xauI

See *xauI* on page A-860 for full details.

The important options of this command are:

TABLE 5-19. xauI Options

Member	Usage
clockType	Determines whether to use an internal or external clock.
podPower	Determines whether 5V power is to be applied to the at pin 4.
userPower	Determines whether 5V power is to be applied to the at pin 5.

Packet over Sonet

The next set of commands allow for the setting of all PoS specific values. If the default values associated with a task are correct, then the corresponding command need not be used. See the *Ixia Reference Guide* for a discussion of SONET/POS load module characteristics.

sonet

See the *Ixia Reference Guide* for a general discussion. See [sonet](#) on page A-657 for full details. The important options of this command are:

TABLE 5-20. sonet Options

Category	Member	Usage
Header	header	Sets the type of Pos header: <ul style="list-style-type: none"> • HDLC ppp: Further settings can be made through the use of hdlc, ppp and pppStatus commands. • Cisco HDLC: Further settings can be made through the use of the hdlc, ppp and pppStatus commands.
Interface	interfaceType	Sets the type and speed of the sonet interface: <ul style="list-style-type: none"> • OC3, OC12 or OC48. • STM1c, STM4c or STM16c.
Transmit	dataScrambling	Controls data scrambling in the sonet framer.
	lineScrambling	Controls line scrambling in the sonet framer.
CRC	rxCrc	Sets the receive CRC mode: 16 or 32 bit mode.
	txCrc	Sets the transmit CRC mode: 16 or 32 bit mode.
APS	apsType	Sets the Automatic Protection Switching mode to linear or ring topology.
	customK1K2	Enables or disables customer K1K2 bytes.
	k1NewState	Allows the K1 byte code value to be sent in the Sonet frame.
Path Signal	k2NewState	Allows the K2 byte code value to be sent in the Sonet frame.
	C2byteExpected	The received path signal label.
	C2byteTransmit	The path signal label to be transmitted.
Error Handling	lineErrorHandler	Enables line error handling.
	pathErrorHandler	Enables path error handling.

Note: The *setDefault* sub-command sets all options at default values, as indicated in *sonet* on page A-657. These values are a consistent setting for an OC12 card and may or may not be appropriate for other cards. In general, the sequence:

```
sonet setDefault
sonet set $chassis $card $port
```

fails.

The *port setFactoryDefaults* command, which relates to a particular port, sets all *sonet* options at default values appropriate for the type of port. The sequence:

```
port setFactoryDefaults $chassis $card $port
sonet set $chassis $card $port
```

always succeeds.

sonetError

This command allows the parameters associated with a variety of simulated SONET errors to be programmed. The errors that are programmed may be inserted once, periodically or continuously. See the *Ixia Reference Guide* for a general discussion. See *sonetError* on page A-674 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-21. sonetError Options

Member	Usage
insertionMode	Controls whether an individual error is inserted periodically or continuously.
errorPeriod errorUnits	The frequency with which periodic errors are inserted, which may be expressed in seconds or frames.
consecutiveErrors	The number of consecutive errors to be inserted at a time.

TABLE 5-22. sonetError Sub-Commands

Member	Usage
setError	Parameters associated with a particular error type are set in IxHal. A <i>set</i> command is needed to get these values into the hardware.
getError	Reads back the values associated with a particular error type into the options described above.
start stop	Starts and stops periodic/continuous error insertion as programmed.
insertError	Inserts a particular error for a single instance. <i>setError</i> and <i>set</i> must be used before this command.

sonetOverhead

This command allows the J0/J1 values of the Sonet overhead to be programmed and read back. See [sonetOverhead](#) on page A-679 for full details. The important options of this command are mentioned in the following table:

TABLE 5-23. sonetOverhead Options

Member	Usage
enableJ0Insertion	Enable the insertion of J0 trace messages.
enableJ1Insertion	Enable the insertion of J1 trace messages.
traceMessageJ0	The J0 trace message, as a hex string.
traceMessageJ1	The J1 trace message, as a hex string.

dcc

This command allows the selection of the DCC byte placement and CRC type. See the *Ixia Reference Guide* for a general discussion. See [dcc](#) on page A-122 for full details. The important options of this command are mentioned in the following table:

TABLE 5-24. dcc Options

Member	Usage
crc	The type of CRC used in the DCC stream
overheadBytes	The placement of the DCC bytes in the line or section overhead bytes.
timeFill	The time fill byte to be used.

RPR

Ixia's Resilient Packet Ring (RPR) implementation is available on selected POS load modules. RPR is a proposed industry standard for MAC Control on Metropolitan Area Networks (MANs) and is defined in IEEE P802.17/D2.1. RPR networks consist of two counter rotating *ringlets*, with nodes called *stations* support MAC clients that exchange data and control information with remote peers on the ring. Up to 255 stations can be supported by RPR networks.

RPR is enabled on a port by selecting the appropriate SONET header encapsulation in the [sonet](#) command:

```
sonet config -header sonetRpr
```

Once enabled, RPR Fairness packets may be set up and transmitted on a regular basis using the [rprFairness](#) command.

For all other RPR messages and encapsulated packets, the [rprRingControl](#) command should be used set up the RPR header.

All IP and ARP packets are automatically encapsulated after the SONET header has been set to RPR. Three commands are used to provide RPR topology discovery, protection and other maintenance:

- *rprProtection*: Provides information related to protection switching, which allows packets to be re-routed or dropped in case of link or station failure.
- *rprOam*: Provides echo and other Operations, Administration and Maintenance (OAM) controls and information.
- *rprTopology*: Provides topology and bandwidth information between nodes to support bandwidth allocation and other functions.

All three message types are added to a stream using their respective *set* sub-commands. A *stream* set command then makes them ready for transmission.

rprRingControl

The *rprRingControl* command is used to set up the content of RPR header used by all RPR packets, except the RPR Fairness Frames, which are set up in the *rprFairness* command. The options are divided into Base Control and Extended Control options. See *rprRingControl* on page A-628 for full details. The important options of this command are mentioned in the following table:

TABLE 5-25. rprRingControl Options

Type	Option	Usage
Base	packetType	Indicates whether the RPR packet is an idle, control, fairness or data frame.
	ringIdentifier	Which ringlet the packet should be transmitted on.
	serviceClass	The class of service that the packet belongs to.
	ttl	The time-to-live for the RPR packet.
	enableFairnessEligible	Whether the packet is eligible for throttling in the fairness algorithm.
	enableOddParity parityBit	Controls parity for Fairness frames.
Extended	enableWrapEnable	Whether the packet is eligible for wrapping in the case of a failure.
	extendedFrame	Indicates that the frame was sent from and to MAC addresses that are not stations. The other fields are then provided as part of the frame.
	ttlBase	The TTL of the original packet prior to encapsulation.
	floodingForm	Whether the frame should be flooded or not and if so to which ringlets.

TABLE 5-25. rprRingControl Options

Type	Option	Usage
	strictOrder	Whether strict ordering on packets should be preserved.
	passedSource	Indicates that a wrapped packet has passed its source.

rprFairness

The *rprFairness* command is used to set up the content of RPR Fairness messages sent periodically from a node. The RPR Fairness Algorithm (FA) is used to manage congestion on the ringlets in an RPR network. Fairness frames are sent periodically to advertise bandwidth usage parameters to other nodes in the network to maintain weighted fair share distribution of bandwidth. The messages are sent in the direction opposite to the data flow; that is, on the other ringlet. See [rprFairness](#) on page A-614 for full details. The important options of this command are mentioned in the following table:

TABLE 5-26. rprFairness Options

Member	Usage
<ring control>	The <i>rprFairness</i> command maintains a separate copy of the Base options discussed in rprRingControl on page 5-17. See that section for details.
enableTransmit	Enables the transmission of RPR Fairness messages.
controlValue	The normalized advertised fair rate value.
messageType	Specifies single or multi-point choke message.
repeatInterval	The frequency of fairness message transmission.
rxAgingInterval	A timeout value for receipt of Fairness messages from other nodes.
rxMacAddress txMaxAddress	The receive and transmit MAC addresses to use in Fairness messages.

rprProtection

The *rprProtection* command is used to build RPR protection messages. Protection messages provide wrapping status information and indicates of a station's desires with respect to wrapping. See [rprProtection](#) on page A-625 for

full details. The important options of this command are mentioned in the following table:

TABLE 5-27. rprProtection Options

Member	Usage
wrapPreferred	A station's ability and/or preference to support wrapping.
jumboPreferred	A station's ability and/or preference to support jumbo frames.
protectionRequestEast	The protection state of the East/West interface.
protectionRequestWest	
sequenceNumber	Use to ensure proper interpretation of Protection messages.
wrappingStatusEast	The wrapping status for traffic received on the East/West interface.
wrappingStatusWest	

rprOam

The **rprOam** command is used to build RPR OAM (Operations, Administration, Management) messages. These messages are sent between stations to determine the operational status of the connection. Following are the types of messages:

- Echo request and response frames: To determine connectivity.
- Flush frames: To prevent mis-ordering of frames.
- Vendor specific frames: For carrying a vendor's OAM information.

See [rprOam](#) on page A-622 for full details. The important options of this command are mentioned in the following table:

TABLE 5-28. rprOam Options

Member	Usage
typeCode	Indicates the type of message: flush, echo request, echo response, or vendor specific.
requestProtectionMode	The requested protection mode for the station.
requestRinglet	Controls which ringlet the receiving station should respond on.
responseProtectionMode	As in <i>requestProtectionMode</i> , but for a response.
responseRinglet	As in <i>requestRinglet</i> , but for a response.
vendorOui	For a vendor specific message, the vendor's OUI designation. The user data for the message should be established using <i>stream</i> background data.

rprTopology

The *rprTopology* command is used to build RPR topology messages. RPR topology messages consist of a set of TLV (type-length-value) settings constructed through the use of the *rprTlvIndividualBandwidth* and *rprTlvBandwidthPair*, *rprTlvWeight*, *rprTlvTotalBandwidth*, *rprTlvNeighborAddress*, *rprTlvStationName*, and *rprTlvVendorSpecific* commands, followed by a call to the *addTlv* command for that type.

A TLV is added to a topology message by configuring the TLV with the appropriate command from the list above and then adding it to the topology message with *rprTopology addTlv type*, where *type* indicates which of the TLVs to use. A TLV may be retrieved from a topology message through the use of *getFirstTlv* / *getNextTlv*. These commands return the **name/pointer** of the command that was used to configure the TLV. This is typically used in the following sequence of commands:

```
set tlvCmd [rprTopology getFirstTlv]
$tlvCmd config ...
```

Each of the TLV commands also has a *type* option which uniquely identifies the type of the TLV.

The individual TLVs are set up using the commands in the following sections. See *rprTopology* on page A-644 for full details. The important sub-commands of this command are mentioned in the following table:

TABLE 5-29. rprTopology Sub-Commands

Member	Usage
addTlv	Adds a TLV to the list associated with the Topology message.
clearAllTlvs	Removes all TLVs in the list.
getFirstTlv	Cycles through the list of TLVs.
getNextTlv	
delTlv	Deletes the currently addressed TLV.

rprTlvIndividualBandwidth and rprTlvBandwidthPair

The *rprTlvIndividualBandwidth* command is used to set up the content of an RPR Individual Bandwidth TLV for use in an RPR topology message. This TLV is added to a topology message by use of the *rprTopology addTlv rprIndividualBandwidth* command.

This command's data is constructed by adding *rprTlvBandwidthPairs*. Bandwidth pairs are constructed through the use of the *rprTlvBandwidthPair* command and then added to this command with the *rprTlvIndividualBandwidth addBandwidthPair* command. Each bandwidth pair corresponds to the reserved bandwidth between this node and a node a number of hops away from this node. The first item in the pair represents the reserved bandwidth on ringlet 0 and the second represents the reserved bandwidth on ringlet 1.

Bandwidth pairs must be added in order; that is, the node one hop away, followed by the node two hops away, etc.

See [rprTlvIndividualBandwidth](#) on page A-632 and [rprTlvBandwidthPair](#) on page A-631 for full details. The important sub-commands of the [rprTlvIndividualBandwidth](#) command are:

TABLE 5-30. rprTlvIndividualBandwidth Sub-Commands

Member	Usage
addBandwidthPair	Adds a TLV to the list associated with the Topology message.
clearAllBandwidthPairs	Removes all TLVs in the list.
getFirstBandwidthPair	Cycles through the list of TLVs.
getNextBandwidthPair	
delBandwidthPair	Deletes the currently addressed TLV.

The important options of the [rprTlvBandwidthPair](#) command are mentioned in the following table:

TABLE 5-31. rprTlvBandwidthPair Options

Member	Usage
bandwidth0	The bandwidth requirements of the two ringlets.
bandwidth1	

rprTlvWeight

The [rprTlvWeight](#) command is used to set up the content of an RPR Weight TLV for use in an RPR topology message. This TLV is added to a topology message by use of the [rprTopology addTlv rprWeight](#) command. See [rprTlvWeight](#) on page A-642 for full details. The important options of the this command are mentioned in the following table:

TABLE 5-32. rprTlvWeight Options

Member	Usage
weightRinglet0	The weight values of the two ringlets.
weightRinglet1	

rprTlvTotalBandwidth

The [rprTlvTotalBandwidth](#) command is used to set up the content of an RPR Total Bandwidth TLV for use in an RPR topology message. This TLV is added to a topology message by use of the [rprTopology addTlv rprTotalBandwidth](#)

command. See [rprTlvTotalBandwidth](#) on page A-638 for full details. The important options of the this command are:

TABLE 5-33. rprTlvTotalBandwidth Options

Member	Usage
bandwidthRinglet0	The total reserved class A0 bandwidth value of the two ringlets.
bandwidthRinglet1	

rprTlvNeighborAddress

The *rprTlvNeighborAddress* command is used to set up the content of an RPR Neighbor Address TLV for use in an RPR topology message. This TLV is added to a topology message by use of the *rprTopology addTlv rprNeighborAddress* command. See [rprTlvNeighborAddress](#) on page A-634 for full details. The important options of the this command are mentioned in the following table:

TABLE 5-34. rprTlvNeighborAddress Options

Member	Usage
neighborMacEast	The total reserved class A0 bandwidth value of the two ringlets.
neighborMacWest	

rprTlvStationName

The *rprTlvStationName* command is used to set up the content of an RPR Station Name TLV for use in an RPR topology message. This TLV is added to a topology message by use of the *rprTopology addTlv rprStationName* command. See [rprTlvStationName](#) on page A-636 for full details. The important options of the this command are mentioned in the following table:

TABLE 5-35. rprTlvStationName Options

Member	Usage
stationName	The name of the station.

rprTlvVendorSpecific

The *rprTlvVendorSpecific* command is used to set up the content of an RPR Vendor Specific TLV for use in an RPR topology message. This TLV is added to a topology message by use of the *rprTopology addTlv rprVendorSpecific* command. See [rprTlvVendorSpecific](#) on page A-640 for full details. The important options of the this command are mentioned in the following table:

TABLE 5-36. rprTlvVendorSpecific Options

Member	Usage
companyId	The IEEE/RAC company identifier.
dependentId	The company dependent part of the identifier
vendorData	The vendor specific data associated with the topology message.

GFP

The Generic Framing Protocol is only available for certain ports, this may be tested through the use of the `port isvalidfeature... portFeatureGfp` command. The GFP framing mode is enabled by setting the `sonet header` option to `sonetGfp`. The GFP header and other options are set in the `gfp` and `gfpOverhead` commands. The `filter` and `filterPallete` commands have access to GFP HEC and CRC error conditions. Additional GFP specific statistics are available in the `stat` command.

`gfp`

The `gfp` command is used to set all GFP framing parameters. The important options of the this command are mentioned in the following table:

TABLE 5-37. `gfp` Options

Member	Usage
enablePli	Set the payload length indicator.
pli	
payloadType	Indicates the type of payload that is encapsulated.
fcs	The type of FCS to include.
channelId	The channel ID for management packets.
coreHecErrors	Controls the inclusion of HEC errors in packets.
typeHecErrors	
extensionHecErrors	

`gfpOverhead`

The `gfpOverhead` command is used to set several operation parameters. The important options of the this command are mentioned in the following table:

TABLE 5-38. `gfpOverhead` Options

Member	Usage
enablePayloadScrambling	Enables payload scrambling.
enableSingleBitErrorCorrection	Enables single bit error correction.
deltaSyncState	Controls state machine transitions.

ppp and pppStatus

`ppp` allows for programming of the Point to Point protocol header, while `pppStatus` can be used to retrieve the current status and values of the PPP negotiation. The options of the two objects are integrated together in the next table. Items from `pppStatus` are indicated in underline mode. See the *Ixia Reference Guide* for a general discussion. See `ppp` on page A-563 and `pppStatus`

on page A-570 for full details. The important options of this command are mentioned in the following table:

TABLE 5-39. ppp/pppStatus Options

Category	Member	Usage
Basic	enable	Enables ppp negotiation.
Negotiation	activeNegotiation	Enables the active negotiation process.
	enableAccmNegotiation	Enables asynchronous control character negotiation.
	enableIip	Enables IP address negotiation
	enableIipV6	Enables IPV6 address negotiation
	enableLqm	Enables line quality monitoring negotiation.
	enableOsi	Enable OSI over PPP negotiation
	enableMpls	Enable MPLS over PPP negotiation
IP Addresses	<u>ipState</u>	The current state of IPCP negotiation
	localIpAddress	The local port's IP address.
	<u>peerIpAddress</u>	The peer's IP address.
IPv6 Interface ID	localIpV6IdType localIpV6Negotiation Mode	The negotiation mode and options.
	<u>ipV6State</u>	The current state of IPV6 CP negotiation
	<u>localIpV6Id</u>	Suggested IPV6 address to be used for the Interface ID.
	localIpV6MacBasedId	Suggested MAC address to be used for the Interface ID.
	peerIpV6IdType peerIpV6Negotiation Mode	The negotiation mode and options.
	<u>peerIpV6Id</u>	Suggested IPV6 address to be used for the Interface ID.
	peerIpV6MacBasedId	Suggested MAC address to be used for the Interface ID.
Retries	configurationRetries	The number of configuration requests to try.
	terminationRetries	The number of termination requests to try.

TABLE 5-39. ppp/pppStatus Options

Category	Member	Usage
Magic Number	<code>useMagicNumber</code> <code>magicNumberNegotiated</code> <code>useMagicNumberRx/Tx</code> <code>rx/txMagicNumberStatus</code>	Enables the use of a magic number in the negotiation to discover looped back connections. The magic number negotiated between the peers. Enable negotiation and use of the magic number in the receive direction/transmit direction. The status and value of transmit and receive magic number negotiation.
Maximum Receive Unit	<code>rxMaxReceiveUnit</code> <code>txMaxReceiveUnit</code>	Maximum frame size in the receive direction. Maximum frame size in the transmit direction.
LQM	<code>lqmReportInterval</code> <code>lqmQualityState</code> <code>lqmReportIntervalRx/Tx</code> <code>lqmReportPacketCounter Rx/Tx</code>	The desired LQM interval to be used during LQM negotiation The current state of the LQM negotiation The negotiation LQM receive/transmit port interval The number of LQM packets received/transmitted
OSI	<code>rxAlignment</code> <code>txAlignment</code> <code>osiState</code> <code>rxAlignment</code> <code>txAlignment</code>	The desired byte alignment for reception/transmission used during negotiation The current state of OSI negotiation The negotiated byte alignment for reception/transmission
MPLS	<code>mplsState</code>	The current state of MPLS negotiation

hdlc

`hdlc` sets the three values associated with the HDLC header. See the *Ixia Reference Guide* for a general discussion. See [hdlc](#) on page A-261 for full details.

The options and sub-commands of this command are mentioned in the following table:

TABLE 5-40. hdlc Options

Member	Usage
address control protocol	The one-byte address field, one-byte control field and two-byte protocol field.

TABLE 5-41. hdlc Sub-Commands

Member	Usage
setCisco	Sets the header variables to the Cisco defaults in IxHal.
setppp	Sets the header variables to the ppp defaults in IxHal.

frameRelay

frameRelay controls Frame Relay specific parameters. sonet config - header must be configured for the correct Frame Relay headers first. See the *Ixia Reference Guide* for a general discussion. See [frameRelay](#) on page A-247 for full details. The values set here are within the Frame Relay header. Note that

stream get must be called before this command's get sub-command. The important options of this command are mentioned in the following table:

TABLE 5-42. frameRelay Options

Member	Usage
addressSize	The address length in the header.
becn	Sets the backward congestion notification bit.
commandResponse	Sets the command or response bit.
control	Sets the control information bit.
discardEligibleBit	Sets the discard eligible bit.
dlci	DLCI core indicator bit in the Frame Relay address field.
counterMode	Parameters used to vary the DLCI between frames.
repeatCount	
maskSelect	
dlciCoreValue	Frame Relay address field.
etherType	The ethernet type of protocol to use.
extensionAddress	Extension address bit 0/1/2/3.
0/1/2/3	
fecn	Sets the forward congestion notification bit.
nlpid	The network layer identifier for the upper-layer protocol.

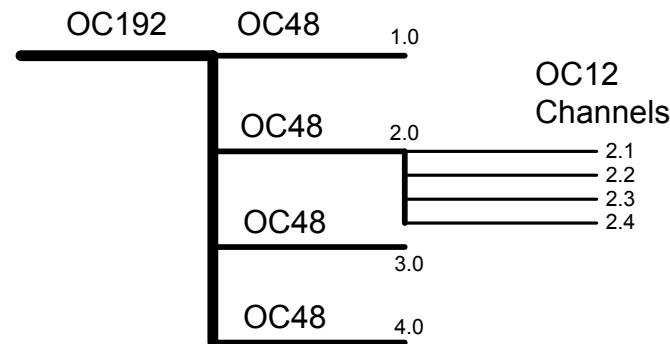
bert and bertErrorGeneration

The *bert* command configures a BERT capable port. The pattern which is transmitted and/or received is programmed. *bertErrorGeneration* is used to insert errors into a transmitted stream. Received errors are available through the use of the *stat* command. See the *Ixia Reference Guide* for a general discussion.

Some of the BERT capable cards support channelized BERT operation. Where available, a *level* parameter indicates which channel is to be controlled. For example, an OC192 channel can be channelized into 4 OC48 channels and each OC48 channel can be channelized into 4 OC12 channels. Each channel selection at each level is represented as a digit in a dot ('.') separated string notation. For

example, the 2nd OC48's 3rd OC12 channel is represented as 2.3. This is illustrated in Figure 5-1.

Figure 5-1. Channelized Bert Label Usage



Note that the OC48 channels may be referred to and operated on using a final .0 digit, as in 3.0.

The important options of the `bert` command are mentioned in the following table:

TABLE 5-43. **bert Options**

Member	Usage
txRxPatternMode	Couples the expected receive pattern with the transmitted, or leaves it independent
txPatternIndex	Determines the transmitted pattern from one of a set or pre-programmed patterns or a user supplied pattern.
txUserPattern	The pattern may be inverted or not.
enableInvertTxPattern	
rxPatternIndex	If the receive pattern is independently programmed
rxUserPattern	from the transmitted pattern, determines the expected
enableInvertRxPattern	receive pattern from one of a set or pre-programmed patterns or a user supplied pattern. The pattern may be inverted or not.

The important options and sub-commands of the `bertErrorGeneration` command are mentioned in the following table:

TABLE 5-44. **bertErrorGeneration Options**

Member	Usage
errorBitRate	Determines the frequency, in bits, with which errors are inserted. The choice may be from a pre-programmed set or set to an arbitrary value.
period	
burstCount	The number of errors inserted at a time.
burstWidth	The number of errors to insert at a time.
burstPeriod	The number of good bits between error insertions.
bitMask	A 32-bit mask indicating which bits within a 32-bit word are to be errored.

TABLE 5-45. bertErrorGeneration Sub-Commands

Member	Usage
startContinuousError	Starts the continuous insertion of programmed errors.
stopContinuousError	Stops the continuous insertion of errors.
insertSingleError	Inserts a single instance of the programmed error.
channelize	Channelizes an OC48 channel down into four OC12 channels. A port must first have been set to channelized mode by setting the <i>port</i> command's <i>transmitMode</i> setting to <i>portTxModeBertChannelized</i> .
isChannelized	Determines whether a level is channelized already.
unChannelize	Unchannelizes an OC48 channel.

bertUnframed

The *bertUnframed* command is used to configure line speed and other operational characteristics for an unframed BERT port. The important options of this command are mentioned in the following table:

TABLE 5-46. bertUnframed Options

Member	Usage
dataRate	The data rate at which data is transmitted.
operation	The type of operation: Normal, diagnostic loopback, or line loopback.

ATM

The next set of commands relates to ATM type cards. See the *Ixia Reference Guide* for a general discussion. Note that different types of ATM encapsulation result in different length headers, as per [Table 5-47](#).

TABLE 5-47. ATM Header Length as a function of Encapsulation

Encapsulation	Header Length
LLC Snap Routed	8
LLC Bridged Ethernet / 802.3	10
LLC Bridged Ethernet / 802.3 No FCS	10
LLC Encapsulated PPP	6
VC Muxed PPP	2
VC Muxed Routed	0
VC Muxed Bridged Ethernet / 802.3	2
VC Muxed Bridged Ethernet / 802.3 No FCS	2

The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in the *dataIntegrity*, *filter*, *flexibleTimestamp*, *ip*, *ipV6Fragment*, *packetGroup*, *protocolOffset*, *qos*, *tableUdfColumn*, *tcp*, *udf*, and *udp* is with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

atmPort

See *atmPort* on page A-35 for full details. The *atmPort* command is used to configure the basic operational characteristics for an ATM port. The important options of this command are mentioned in the following table:

TABLE 5-48. atmPort Options

Member	Usage
interfaceType	Sets the port to UNI (user-network interface) or NNI (network-network interface).
enableCoset	Enables or disables the use of the Coset algorithm with the Header Error Control byte.
enablePattern Matching	Enables or disables the use of the <i>atmFilter</i> command to control capture and statistics. Enabling this feature reduces the maximum number of VCCs that may be used from 16K to 12K.
fillerCell	Designates which of two types of cells is transmitted during idle periods: <ul style="list-style-type: none"> • Idle Cell (VPI/VCI = 0 and CLP = 1) • Unassigned Cell (VPI/VCI = 0 and CLP = 0)
packetDecodeMode	The mode in which to decode received packets: Frame or cell.
reassemblyTimeout	The period of time to wait for a cell on a channel.
sourceLocationId	The source location ID.

atmHeader

See [atmHeader](#) on page A-12 for full details. The *atmHeader* command is used to configure the 5-byte ATM header inserted in packets within streams. Note that *stream get* must be called before this command's *get* sub-command. The important options of this command are mentioned in the following table:

TABLE 5-49. atmHeader Options

Member	Usage
vpi/vci enableAutoVpiVci Selection	Sets the Virtual Path Identifier (VPI) and Virtual Circuit Identifier (VCI) for the header. The <i>enableAutoVpiVciSelection</i> control sets these to 0/32.
genericFlowControl	The Generic Flow Control value, used for device control signalling.
enableCL	Controls the congestion loss bit of the payload type.
cellLossPriority	The cell's priority, when cells must be dropped. A value of 0 has a higher priority than 1.
hecErrors	The number of bit errors to insert in the HEC byte.
encapsulation	The type of ATM encapsulation to be used.
aal5Error	Force the insertion of AAL5 errors.
enableCpcsLength cpcsLength	Allows the CPCS PDU length to be set.
header	A read-only 5-byte header value, set from the other options.

atmHeaderCounter

See [atmHeaderCounter](#) on page A-15 for full details. The *atmHeaderCounter* command is used to configure the counter parameters that allow the value of the ATM header's VPI and VCI fields to vary. The VPI and VCI values are separately controlled using the same command. Following are the types of counters available:

- Fixed: A single value is used throughout.
- Counter: An incrementing counter is applied.
- Random: A masked set of bits are randomly set.
- Table: A table of values is repetitively used.

The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-50. atmHeaderCounter Options

Member	Usage
type	The type of counter used: Fixed, counter, random, or table.
mode	If a <i>counter</i> type is used, then this indicates whether the counter counts up or down continuously or for a particular count. The step size is also specified.
repeatCount	
step	
maskselect	If the <i>random</i> type is used, this indicates which bits of the value are fixed and to what values.
maskvalue	
dataItemList	If the <i>table</i> type is used, then this is the table of values to be used round-robin.

TABLE 5-51. atmHeaderCounter Sub-Commands

Member	Usage
set	Sets the options for either the VPI or VCI value.
get	Gets the options for either the VPI or VCI value.

atmOam

The **atmOam** command is used to configure multiple ATM OAM messages to be transmitted on an ATM port. The basic parameters for all OAM messages are configured in the options of this command. Additional parameters that are particular to a specific OAM message are taken from the following additional commands: *atmOamActDeact*, *atmOamAis*, *atmOamFaultManagementCC*, *atmOamFaultManagementLB* or *atmOamRdi*.

Once configured, the OAM message for a VPI/VCI pair is added to the list associated with this command with the *add* sub-command. Transmission of the OAM messages is initiated with the *start* sub-command and stopped with the *stop* sub-command.

Trace information, if enabled with the *enableTrace* option is retrieved using the *atmOamTrace* command.

Refer to *atmOam* on page A-18 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-52. atmOam Options

Member	Usage
enableTx	Enable the use of this list element.
enableCC	Enable continuous checking.
enableLB	Enable loopback.
enableTrace	Enable collection of trace messages for the VPI/VCI.
functionType	The type of the OAM message.

TABLE 5-52. atmOam Options

Member	Usage
endPointsType	The type of endpoints: Segment or end-to-end.
cellFlowType	The cell flow type: F4 or F5.

TABLE 5-53. atmOam Sub-Commands

Member	Usage
select	Select the particular port to operate on.
add	Add an OAM message to the list for a particular VPI/VCI pair.
removeAll del	Delete all or a particular list entry.
getFirstPair getNextPair	Iterate through the list entries.
start stop	Start and stop the transmission and reception of OAM messages.

atmOamActDeact

The *atmOamActDeact* command holds command specific options for the activation/deactivation message. Refer to *atmOamActDeact* on page A-25 for full details. The important options of this command are mentioned in the following table:

TABLE 5-54. atmOamActDeact Options

Member	Usage
enableTxContinuous txCount	Controls continuous or counted message transmission.
action	The direction of the action: One way or bi-directional.
correlationTag	The correlation tag.
defectLocation	The defect location.
messageId	The particular message: Activate/Deactivate Confirmed/Denied.

atmOamAis

The *atmOamAis* command holds command specific options for the AIS message. Refer to *atmOamAis* on page A-27 for full details. The important options of this command are mentioned in the following table:

TABLE 5-55. atmOamAis Options

Member	Usage
enableTxContinuous txCount	Controls continuous or counted message transmission.

atmOamFaultManagementCC

The *atmOamFaultManagementCC* command holds command specific options for the Fault Management Continuous Checking message. Refer to *atmOamFaultManagementCC* on page A-28 for full details. The important options of this command are mentioned in the following table:

TABLE 5-56. atmOamFaultManagementCC Options

Member	Usage
enableTxContinuous txCount	Controls continuous or counted message transmission.

atmOamFaultManagementLB

The *atmOamFaultManagementLB* command holds command specific options for the Fault Management Loopback message. Refer to *atmOamFaultManagementLB* on page A-29 for full details. The important options of this command are mentioned in the following table:

TABLE 5-57. atmOamFaultManagementLB Options

Member	Usage
enableTxContinuous txCount	Controls continuous or counted message transmission.
correlationTag	The correlation tag.
loopbackIndication	The loopback indication: Reply or request.
loopbackIndicationId	The loopback indication ID.
sourceLocationId	The source location ID.

atmOamRdi

The *atmOamRDI* command holds command specific options for the RDI message. Refer to *atmOamRdi* on page A-31 for full details. The important options of this command are mentioned in the following table:

TABLE 5-58. atmOamRdi Options

Member	Usage
enableTxContinuous txCount	Controls continuous or counted message transmission.
defectLocation	The defect location.

atmOamTrace

The **atmOamTrace** command is used to retrieve ATM OAM messages. These are collected for any OAM message in which the *enableTrace* option was set to true when *atmOam add* was called.

Messages are collected into a circular buffer of *maxNumTrace* messages in size. Newest entries replace oldest entries as necessary. The get chassis card port sub-

command is used to retrieve all of the message. The other *get* commands are used to look at particular entries.

Refer to [atmOamTrace](#) on page A-32 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-59. atmOamTrace Options

Member	Usage
maxNumTrace	The maximum number of traces to hold in the buffer.
numTrace	The number of traces in the buffer.
traceString	The entire trace message as a single string.
functionType	The components of the message.
timeStamp	
txRxType	
vci	
vpi	

TABLE 5-60. atmOamTrace Sub-Commands

Member	Usage
get ch card port	Get the trace messages.
get index	Get a particular trace message.
clear	Clear the message buffer.
getFirst	Iterate through the messages.
getNext	

Circuit

The following commands support the Virtual Concatenation feature for 2.5G and 10G MSM cards.

sonetCircuit

The *sonetCircuit* command holds all the circuits. Refer to [sonetCircuit](#) on page A-663 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-61. sonetCircuit Options

Member	Usage
txActiveTimeslot List	Configure the active tx members. (<i>default</i> = "")
rxActiveTimeslot List	Configure the active rx members. (<i>default</i> = "")
txIdleTimeslotList	Configure the idle tx members. (<i>default</i> = "")
rxIdleTimeslotList	
name	Configure the circuit name. (<i>default</i> = "")

TABLE 5-61. sonetCircuit Options

Member	Usage
direction	Configure the direction. (<i>default = circuitBidirectionSymmetrical</i>)
txType	Configure the tx payload speed. (<i>default = circuitPayloadRateSTS1mv</i>)
rxType	Configure the rx payload speed. (<i>default = circuitPayloadRateSTS1mv</i>)
enableTxLcas	Enable the Lcas on transmit side. (<i>default = FALSE</i>)
enableRxLcas	Enable the Lcas on receive side. (<i>default = FALSE</i>)
index	Read only.This parameter is used to view the circuit index assigned by hardware. (<i>default = 0</i>)

TABLE 5-62. sonetCircuit Sub-Commands

Member	Usage
cget option	Returns the current value of the configuration option given by <i>option</i> .
config option value	Modify the configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.
setDefault	Sets to IxTclHal default values for all configuration options.

sonetCircuitList

The *sonetCircuitList* command holds all the circuits. Refer to [sonetCircuitList](#) on page A-665 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-63. sonetCircuitList Options

Member	Usage
numCircuits	Read only. This parameter is used to display the number of existing circuits in the circuit list.

TABLE 5-64. sonetCircuitList Sub-Commands

Member	Usage
add	Adds a new circuit and verifies that the circuit can be added.
cget option	Returns the current value of the configuration option given by <i>option</i> .
clearAllCircuits	Remove all the circuits from the Sonet circuit list.
config option value	Modify the configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

TABLE 5-64. sonetCircuitList Sub-Commands

Member	Usage
del <i>circuitID</i>	Deletes the circuit with the given ID.
get <i>circuitID</i>	Gets the existing circuit with the given circuit ID.
getFirst	Gets the first circuit from the Sonet circuit list.
getNext	Gets the next circuit from the Sonet circuit list.
select <i>chasID cardID portID</i>	Select the port where the IxTclHal configurations is set to local IxHal.
set <i>circuitID</i>	Modify the existing circuit with the given circuit ID.
setDefault	Sets to IxTclHal default values for all configuration options.

sonetCircuitProperties

The *sonetCircuitProperties* command is used to configure circuit properties after the circuit is added. The Sonet properties for the circuit is configured here.

Refer to [sonetCircuitProperties](#) on page A-672 for full details. The important options of this command are mentioned in the following table:

TABLE 5-65. sonetCircuitProperties Options

Member	Usage
transmitMode	This parameter is used to configure the transmit mode. (<i>default = circuitTxModePacketStreams</i>)
payloadType	This parameter is used to configure the Sonet header payload type. (<i>default = sonetHdlcPppIp</i>)
dataScrambling	This parameter is used to configure the Sonet dataScrambling payload type. (<i>default = false</i>)
C2byteTransmit	This parameter is used to configure the Sonet C2byteTransmit . (<i>default = 22</i>)
C2byteExpected	This parameter is used to configure the Sonet C2byteExpected. (<i>default = 22</i>)
rxCrc	Used to configure Rx CRC.
txCrc	Used to configure Rx CRC.
index	Read only. This parameter is used identify the circuit with associated ID.

TABLE 5-66. sonetCircuitProperties Sub-Commands

Member	Usage
cget option	Returns the current value of the configuration option given by <i>option</i> .
config option value	Modify the configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.
get chassis ID cardID portID circuitID	Gets the existing circuit properties with the given circuit ID.
set chassis ID cardID portID circuitID	Modify the existing circuit properties with the given circuit ID.
setDefault	Sets to IxTclHal default values for all configuration options.

lcas

The *lcas* command sets up LCAS configuration for receive and transmit. This enables configuring the LCAS debug/trace messages. Refer to [lcas](#) on page A-389 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-67. lcas Options

Member	Usage
rsAck	Configure the timeout value for Rs_Ack(s) for Rx Lcas. (<i>default</i> = 10)
holdOff	Configure the hold off timeout for Rx Lcas. (<i>default</i> = 10)
waitToRestore	Configure the wait to restore timeout for the Rx Lcas. (<i>default</i> = 10)

TABLE 5-68. lcas Sub-Commands

Member	Usage
cget option	Returns the current value of the configuration option given by <i>option</i> .
config option value	Modify the configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.
get chassis ID cardID portID circuitID	Gets Lcas information for the existing circuit with the given circuit ID.
set chassis ID cardID portID circuitID	Modify Lcas information for the existing circuit with the given circuit ID, if Lcas is enabled in sonetCircuit.
setDefault	Sets to IxTclHal default values for all configuration options..

10GE

Link Fault Signaling

Link fault signalling is implemented on several of the 10GE cards. In addition to several additional statistics which track local and remote faults, the link fault signalling implementation allows local and remote faults to be inserted into transmissions. The following commands are used to control link fault signalling:

- *linkFaultSignaling*: Control over fault insertion
- *customOrderedSet*: Definition of custom fault insertion signals

linkFaultSignaling

The *linkFaultSignaling* command is used to a series or continuous stream of link fault signals. The series/stream consists of good and bad period, where the bad periods may send local, remote or custom errors. Errors are called ordered sets; two, named A and B, are available for insertion. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-69. linkFaultSignaling Options

Member	Usage
enableLoopContinuously	Enables continuous looping or sets a count for a series.
loopCount	
sendSetsMode	Specifies whether ordered set A and/or B is used in the error insertion.
orderedSetTypeA	For each of the two ordered set types, indicates whether the type should insert a local error, a remote error or a custom ordered set. Custom ordered sets are defined through the use of the <i>customOrderedSet</i> command.
orderedSetTypeB	
contiguousErrorBlocks	The number of contiguous 66-bit blocks with errors to insert.
contiguousGoodBlocks	The number of contiguous 66-bit blocks without errors to insert between bad blocks.

TABLE 5-70. linkFaultSignaling Sub-Commands

Member	Usage
startErrorInsertion	Starts the error insertion process.
stopErrorInsertion	Stops the error insertion process.

customOrderedSet

The *customOrderedSet* command is used to define the byte-by-byte contents of a link fault signaling error message. Two sets are maintained: set type A and B. The important options of this command are mentioned in the following table:

TABLE 5-71. customOrderedSet Options

Member	Usage
blockType	The type of the error block.
syncBits	The value of the sync bits.
byte1–byte7	The contents of the remaining bytes in the 66-bit block.

txRxPreamble

The *txRxPreamble* command is used to set the options related to preamble transmit and receive operation on 10GE LAN ports. The important options of this command are mentioned in the following table:

TABLE 5-72. txRxPreamble Options

Member	Usage
rxMode	The receive mode for the port: SFD detect, by byte count, or the same as <i>txMode</i> .
txMode	The transmit mode for the port: SFD detect or by byte count.
enableCiscoCDL	Enables the use of Cisco CDL headers instead of the Ethernet header.
enableCDLStats	Enables the collection of CDL statistics and capture.
enablePreambleView	Enables the inclusion of preamble data in the <i>stream</i> packetview.

Optical Digital Wrapper / FEC

The optical digital wrapper provides for generic framing as specified in ITU-T G.709. At the present time, only FEC error insertion is enabled.

`opticalDigitalWrapper`

This command enables the use of the wrapper. The important options of this command are mentioned in the following table:

TABLE 5-73. `opticalDigitalWrapper` Options

Member	Usage
<code>enableFec</code>	Enables the use of the wrapper and FEC.
<code>enableStuffing</code>	Enables the use of stuffing.
<code>payloadType</code>	The emulated payload type.

`fecError`

Forward Error Correction (FEC) is a method of communicating data that corrects errors in transmission on the receiving end. Prior to transmission, the data is put through a predetermined algorithm that adds extra bits specifically for error correction to any character or code block. If the transmission is received in error, the correction bits are used to check and repair the data. This feature is only available for certain port types; this may be tested through the use of the `portisValidFeature... portFeatureFec` command. FEC insertion must be enabled through the use of the *Optical Digital Wrapper / FEC* command.

The `fecError` command allows you to inject FEC errors into transmitted data. The following modes are controlled by the `injectionMode` option:

- Single: A single instance of an error is inserted.
- Rate: Errors are inserted at one of a set of pre-determined rates as controlled by the `errorRate` option.
- Burst: Continuous bursts of errors is inserted as determined by the `subrow`, `burstSize`, `offset`, `errorBits` and `numberOfRowsToSkip` options.

Single errors are inserted with the `injectError` sub-command and the `start` and `stop` commands are used to start and stop rate and burst error insertion. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-74. `fecError` Options

Member	Usage
<code>injectionMode</code>	Indicates whether a single error, error rate or burst is to be inserted.
<code>errorRate</code>	Indicates the continuous error rate when rate error insertion is used.

TABLE 5-74. fecError Options

Member	Usage
burstSize	Options which control the insertion of error bursts.
errorBits	
numberOfRowsToSkip	
offset	
subrow	

TABLE 5-75. fecError Sub-Commands

Member	Usage
start	Starts the error insertion process for rate and burst insertion modes.
stop	Stop the error insertion process for rate and burst insertion modes.
injectError	Injects a single error, when the injection mode is set to single error.

CDL Support

Cisco Converged Data Layer (CDL) support is enabled through the use of the [*txRxPreamble enableCiscoCDL*](#) option. When this option is set the [*cdlPreamble*](#) command is used to set up the CDL preamble. The [*txRxPreamble enableCDLStats*](#) option controls the collection of CDL statistics and preamble capture. The [*txRxPreamble enablePreambleView*](#) option controls the format of the [*stream packetView*](#). When CDL mode is active, the [*filter*](#) command is able to filter on CDL header errors.

cdlPreamble

The [*cdlPreamble*](#) command configures the CDL Preamble that is enabled through the use of the [*txRxPreamble enableCiscoCDL*](#) option. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-76. cdlPreamble Options

Member	Usage
oam	Packet type and OAM information
messageChannel	The in-band message channel
applicationSpecific	Application specific data
enableHeaderCrcOverwrite headerCrc	Allows the precalculated header to be overridden.
startOfFrame cdlHeader	Read-only reflections of the start of frame byte and the entire CDL preamble.

TABLE 5-77. **cdlPreamble Sub-Commands**

Member	Usage
decode	Decodes a captured frame.

xfp

UNIPHY-XFP cards have two additional options that control monitoring of LOS (Loss of Signal) and module ready status. See [xfp](#) on page A-862 for full details. The important options of this command are mentioned in the following table:

TABLE 5-78. **xfp Options**

Member	Usage
enableMonitorLos	Enables the port to monitor Loss of Signal. In this case, the Loss of Signal status is used to determine Link State.
enableMonitorModuleReady	Enables the port to monitor whether the module is ready. In this case, no transmit, received or statistics operations are performed until the module is ready.

lasi

10GE XENPAK cards have an additional link alarm status interrupt (LASI) set of registers which control the interrupt operation. See [lasi](#) on page A-385 for full details. The important options of this command are mentioned in the following table:

TABLE 5-79. **lasi Options**

Member	Usage
enableMonitoring	Enables the monitoring of the LASI status registers so as to clear the interrupt signal.
enableAutoDetectedOUIDeviceAddress	Enables the automatic detection of a devices OUI address.
ouiDeviceAddress	The OUI device address of the LASI status registers.
controlRegister	The values of the registers which control LASI operation.
rxAlarmControlRegister	
txAlarmControlRegister	

Power Over Ethernet

The Power over Ethernet (PoE) ports are controlled by the following commands:

- [poePoweredDevice](#) - sets up and applies voltage and current to emulate a PoE powered device.
- [poeAutoCalibration](#) - sets up and controls port calibration.
- [poeSignalAcquisition](#) - sets up and controls the ability to measure time and amplitude values on the PoE signal.

poePoweredDevice

The *poePoweredDevice* command is used to setup the parameters by which a PoE Powered Device (PD) is emulated on a port. The port can emulate a device that uses either *Alternative A* and/or *Alternative B*. This is controlled by the *relayControl* option. The emulated class is controlled by the *enableClassSignature* and *signatureValue* options; the *classType* indicates the calculated class based on the signature value. The emulated detection signature is controlled by the *enableDetectionSignature*, *rsig*, *csig* and *enableCsig10uF* options. The emulated Alternating Current Maintain Power Signature (ACMPS) is controlled by the *enableAcMpsSignature*, *rpd*, *cpd* and *enable CpdAdd10uF* options.

Once the PSE (Power Sourcing Equipment) has classified the emulated PD, it should provide power to the port. The power requirements of the emulated port are controlled by the *steadyStateLoadControl*, *controlledCurrent*, *controlledPower* and *idleCurrent* options. Transient load variations may be inserted through the use of the *enableTransientLoadControl*, *transientLoadControl*, *pulseWidth*, *duty*, *pulsedCurrent* and *slewRate* options. Pulses are applied through the use of the *portGroup setCommand* sub-command, with an *loadPoEPulse* value or through the high-level *ixLoadPoePulse* and *ixLoadPortPoePulse* commands; if *enableTransientLoadControl* is *true* and *transientLoadControl* is set to *poeLoadControlSinglePusle*, then a pulsed current as indicated by *pulsedCurrent* and *slewRate* is injected for the period indicated by *pulseWidth*.

The voltage thresholds that are used by the PD to detect state transitions may be set by the *vOperate*, *vOff*, *vClassify*, *vDetect* and *vNoop* options. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-80. **poePoweredDevice Options**

Member	Usage
<i>relayControl</i>	Indicates which device Alternatives should be emulated.
<i>enableClassSignature</i> <i>signatureValue</i> <i>classType</i>	Control the setting of the class signature.
<i>enableDetectionSignature</i> <i>rsig</i> <i>csig</i> <i>enableCsigAdd10uF</i>	Controls the setting of the detection signature.
<i>enableAcMpsSignature</i> <i>rpd</i> <i>rpdRangeControl</i> <i>cpd</i> <i>enableCpdAdd10uF</i>	Controls the setting of the ACMPS signature.

TABLE 5-80. poePoweredDevice Options

Member	Usage
steadyStateLoadControl	Controls the steady state power requirements.
controlledCurrent	
controlledPower	
idleCurrent	
enablePulseOnStart	Controls the application of transient loads.
enableTransientLoadControl	
transientLoadControl	
pulseWidth	
duty	
pulsedCurrent	
slewRate	
vOperate	Controls the state transition voltage thresholds.
vOff	
vClassify	
vDetect	
vNoop	

TABLE 5-81. poePoweredDevice Sub-Commands

Member	Usage
setNominal	Sets nominal values for all types.

poeAutoCalibration

The *poePoweredDevice* command is used to initiate a PoE port calibration and/or determine the status of a calibration. Calibration of all PoE ports is performed at chassis power-up time.

A calibration is initiated by calling the *initiateCalibrate* sub-command. The calibration may take up to 20 seconds. The results of a calibration, either while it is preceding or after it has completed, can be determined by first calling *requestStatus*, waiting a second and then calling *get*. The status of the calibration is then available through the options in this command. Refer to [poeAutoCalibration](#) on page A-508 for complete details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-82. poeAutoCalibration Options

Member	Usage
currentReadbackStatus	The current status of the auto-calibration for each item: One of testing, pass or fail.
voltageReadbackStatus	
iClassRangeStatus	
iLoadRangeStatus	
iPulseRangeStatus	

TABLE 5-83. poeAutoCalibration Sub-Commands

Member	Usage
initiateCalibrate	Starts the calibration process.
requestStatus	Requests the status of the calibration.
get	Reads back the status of the calibration so that it may be read with <i>cget</i> 's.

poeSignalAcquisition

The *poeSignalAcquisition* command is used to set up and capture the time between two signal transition events. The amplitude of the a signal may also be measured a fixed time after the first signal transition.

The *startTriggerSource*, *startTriggerSlope* and *startTriggerValue* are used to indicate the signal to be used for the first event, the slope that it should transition (positive or negative) and the value that should be matched. Similarly, the *stopTriggerSource*, *stopTriggerSlope* and *stopTriggerValue* are used to indicate the signal to be used for the second event. The *enableTime*, *enableAmplitude* and *amplitudeMeasurementDelay* options are used to condition the measurements made.

Arming of the signal acquisition is accomplished through the use of the *portGroup* command with the *armPoeTrigger* value, or the *ixArmPoeTrigger* and *ixArmPortPoeTrigger* high-level commands. The arming may be aborted through the use of the *portGroup* command with the *abortPoeTrigger* value, or the *ixAbortPoeArm* and *ixAbortPortPoeArm* high-level commands.

A number of statistics available through the *stat*, *statGroup*, *statList* and *statWatch* commands support operation of this command. The status of the arming may be read from the *statPoeTimeArmStatus* and *statPoeAmplitudeArmStatus* options. The status of the triggering may be read from the *statPoeTimeDoneStatus* and *statPoEAmplitudeDoneStatus* options. The time and amplitude values are visible in the *statPoeMonitorTime* and *statPoeMonitorAmplitudeDCVolts* and *statPoeMonitorAmplitudeDCAmps* options after a trigger has completed.

portGroup

Port groups provide a means of creating a group of ports on which an action may be performed or command may be sent. A single instance of portGroup may be used to maintain a number of groups. See [portGroup](#) on page A-557 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-84. portGroup Options

Member	Usage
lastTimestamp	The timestamp, a 64-bit number of nanoseconds, of when the last command was sent to the hardware as a result of a setCommand method execution.

TABLE 5-85. portGroup Sub-Commands

Member	Usage
create	Creates a new port group, identified by a unique number.
destroy	Destroys a port group.
add	Adds a port to a port group.
del	Deletes a port from a port group.
canUse	Tests to see whether the current user can use the ports in a group. That is, whether you own the ports or ports are being used by someone else.
setCommand	<p>Performs an action or sends a command to all of the ports in a group.</p> <p>Transmit commands:</p> <ul style="list-style-type: none"> • Start / stop • Staggered start • Pause • Step • Clear time stamp <p>Receive:</p> <ul style="list-style-type: none"> • Start / stop capture • Reset statistics • Start / stop latency • Clear latency <p>Protocols:</p> <ul style="list-style-type: none"> • Start/stop each of the protocols <p>Others:</p> <ul style="list-style-type: none"> • Take / clear ownership • Force take / clear ownership
clear/set Scheduled TransmitTime	Clears or sets the maximum amount of time that a group of ports transmits. This is only valid for ports that support the <i>portFeatureScheduledTxDuration</i> feature, which may be tested with the port isValidFeature command.

TABLE 5-85. **portGroup** Sub-Commands

Member	Usage
write	Sends port properties such as speed, duplex mode and autonegotiation to the hardware. All other values may be sent with <code>writeConfig</code> .
writeConfig	Sends streams, filter and capture parameters to the hardware.

Data Transmission

Streams and Flows

Streams and flows are the means by which data is applied to the DUT. Streams are generated ‘on the fly’ by the Ixia hardware. Flows are data arrays located on disk and associated with a port. Multiple streams are defined and associated with a port through the use of the *stream* command. *stream* provides for the transitions between streams, gaps, addressing and basic frame control. See the *Ixia Reference Manual* for a general discussion. The following additional commands are required for further packet header and data contents:

- *stream*: Construct streams.
- *streamRegion*: Controls common stream properties.
- *weightedRandomFramesize*: Advanced weighted random distributions of frame sizes.
- *flexibleTimestamp*: Place the time stamp at different locations.
- *udf*: User Defined Fields, algorithmically or manually generated.
- *tableUdf*: Table driven UDF.
- *tcpRoundTripFlows*: Generate packets for round trip flow analysis.
- *packetGroup*: Generate data for packet group latency measurements.
- *dataIntegrity*: Generate additional data integrity values.
- *Sequence Checking*: Generate data for additional sequence checking.
- *forcedCollisions*: Generate deliberate collisions.
- *protocol*: Establish basic protocol parameters.
- *protocolOffset*: Change the location of protocol headers in a packet.
- *isl*: Set up header parameters for Cisco ISL.
- *vlan*: Set up header parameters for VLANs.
- *stackedVlan*: Set up stacked VLANs (Q-in-Q).
- *mpls* and *mplsLabel*: Generate MPLS headers and control messages.
- *The ipx command provides for the setting of IPX header elements. See ipx on page A-375 for full details. Note that stream get must be called before this command's get sub-command. The important options of this command are mentioned in the following table.* Set up IPX header parameters.
- *The name of the associated protocol object must be set to 'ip' and the appName must be set to 'arp' to successfully use this command. The arp command allows ARP packets to be constructed. See arp on page A-2 for full details.* Generate ARP messages.
- *IP*: Set up IPv4 header parameters.
- *IPv6*: Set up IPv6 header parameters.
- *tcp*: Set up TCP/IP header parameters.
- *udp*: Set up UDP/IP header parameters.
- *igmp*: Generate IGMP messages.

- *icmp*: Generate ICMP messages.
- *rip* and *ripRoute*: Generate RIP messages.
- *dhcp*: Generate DHCP messages.
- *pauseControl*: Generate pause control packet.
- *srpArp*: Generate an SRP ARP packet.
- *srpIps*: Generate an SRP IPS (Intelligent Protection Switch) packet.
- *srpDiscovery*: Generate an SRP Discovery packet.
- *srpMacBinding*: Configure MAC bindings for Discovery packets.
- *srpUsage*: Set up periodic SRP Usage packet generation.
- *streamQueueList*: Manages the stream queues associated with ATM ports.
- *streamQueue*: For ATM ports, manages the streams in a stream queue.
- *streamQueue*: For ATM ports, manages the streams in a stream queue.
- *npiV*: To configure an unconnected NPIV interface.

stream

`stream` controls the basic structure of streams: stream to stream transition, inter-stream/frame/burst gaps, and addressing. It also controls the common frame contents: size, base data pattern, checksum, and identity record. The other commands in this section may be used for specific protocols and header data. Multiple streams may be created and connected to each other through the use of their IDs. See the *Ixia Reference Guide* for a general discussion. See *stream* on page A-747 for full details.

Some ports support the scheduled transmit duration feature, which may be tested with the *port isValidFeature* command. This feature allows a group of ports to run for a fixed number of seconds, or for the period indicated by its streams, whichever comes first. This feature is controlled with the *portGroup setScheduledTransmitTime* and *clearScheduledTransmitTime* commands.

The *setQueue* and *getQueue* sub-commands are used to set up streams within ATM ports. Queues are discussed in *streamQueueList* and *streamQueue*. ATM streams may have incrementing and/or random frame sizes, but only 16 of either type. All other streams are forced to fixed size.

Note: To modify the options of any command that contributes to the configuration of streams, it is always necessary to perform a *stream get chassis card port stream* command.

The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-86. stream Options

Category	Member	Usage
Stream Control	enable	This stream is enabled or not. Disabled streams are skipped during transmission.
	dma	<p>The type of stream and relationship to another stream:</p> <ul style="list-style-type: none"> • Continuous packet • Continuous burst • Stop after stream • Advance to next stream • Return to stream ID (labeled as <code>gotoFirst</code> for historical reasons) • Loop to stream ID (labeled as <code>firstLoopCount</code> for historical reasons)
	returnTold	The stream ID to return to for the return to and loop to stream ID <code>dma</code> types.
	loopCount	This is the repeat count for the <code>dma</code> choice 'loop to stream ID'.
	numBursts	The number of bursts, ignored for <code>dma</code> choices continuous packet and continuous burst.
	numFrames	The number of maximum frames in a stream, ignored for <code>dma</code> choice continuous packet.
	priorityGroup	Specifies the priority group of the stream.
Inter-Frame Gap	gapUnit	<p>The choice of units for <code>ifg</code>, <code>isg</code> and <code>ibg</code>. The choices are mentioned as follows:</p> <ul style="list-style-type: none"> • Nano-seconds • Micro-seconds • Milli-seconds • Seconds • Clock ticks that vary with the port type
	ifgType	Indicates whether the inter-frame gap is a fixed value or random between a minimum (<code>ifgMIN</code>) and maximum value (<code>ifgMAX</code>).
	ifg	The inter-frame gap expressed in <code>gapUnit</code> units.

TABLE 5-86. stream Options

Category	Member	Usage
	ifgMIN	The minimum gap generated for ifgType of random.
	ifgMAX	The maximum gap generated for ifgType of random.
	rateMode	Indicates whether to use the ifg value, percentage of the maximum transmission rate, frames per second or bits per second.
	percentPacketRate	If rateMode indicates, then ifg is calculated based on a desired percentage of maximum transmission rate.
	fpsRate	If indicated by rateMode, the desired frames per second.
	bpsRate	If indicated by rateMode, the desired bits per second.
	framerate	(Read-only) The actual rate, in frames per second that the stream transmits at.
Inter-Burst Gap	enableIbg	Enables the use of inter-burst gaps.
	ibg	The inter-burst gap expressed in gapUnit units.
Inter-Stream Gap	enableIsbg	Enables the use of inter-stream gaps.
	isbg	The inter-stream gap expressed in gapUnit units.
Addressing	da	First destination MAC address assigned to the stream.
	daMaskValue/ daMaskSelect	Indicates which bits of the destination MAC address are to be manipulated and their initial values.
	numDA	The number of destination MAC addresses that is used.
	daRepeatCounter	Indicates how the destination MAC address is to be incremented or decremented from packet to packet.
	sa	First source MAC address assigned to the stream.

TABLE 5-86. stream Options

Category	Member	Usage
	saMaskValue/ saMaskSelect	Indicates which bits of the source MAC address are to be manipulated and their initial values.
	numSA	The number of source MAC addresses that is used.
	saRepeatCounter	Indicates how the source MAC address is to be incremented or decremented from packet to packet.
	enableSourceInterface	Enables the use of the MAC and IP addresses from the interface table in lieu of the <i>sa</i> value.
	sourceInterfaceDescription	The name of the interface on the port to use.
Frame Control	frameSizeType	<p>The type of frame size calculation:</p> <ul style="list-style-type: none"> • Fixed size as indicated in <code>framesize</code>. • Random size between <code>frameSizeMin</code> and <code>frameSizeMax</code>. Some modules support more advanced random frame sizes, see <i>weightedRandomFramesize</i>. • Incrementing packet to packet. • Automatically calculated, depending on protocol dependent contents.
	framesize	The size of all frames if <code>frameSizeType</code> is fixed.
	frameSizeMin	The minimum frame size if <code>frameSizeType</code> is random.
	frameSizeMax	The maximum frame size if <code>frameSizeType</code> is random.
	frameType	The type field of the Ethernet frame.
	preambleSize	Number of bytes in the frame preamble.
	patternType	<p>Dictates the type of data pattern manipulation:</p> <ul style="list-style-type: none"> • increment/decrement bytes or words • random data • fixed repeating pattern chosen from <code>dataPattern</code> • fixed non-repeating pattern chosen from <code>dataPattern</code>

TABLE 5-86. stream Options

Category	Member	Usage
	dataPattern	One of a number of fixed patterns of data, including all 1's and all 0's, plus a choice for a user specified pattern.
	pattern	If dataPattern indicates a user specified pattern, this string specifies the contents.
	fcs	The type of FCS error to be inserted into the frame (or none).
	enableTimestamp	Whether to insert a Frame Identity Record into the last 6 bytes of the packet.
Misc	asyncIntEnable	Allow asynchronous interrupts required by the protocol server.
	packetView	(Read-only) Shows the packets that are about to be transmitted. If the port's <code>port.transmitMode</code> is set to <code>portTxPacketFlows</code> , then this displays all of the packets to be transmitted. This data may be saved and used to specify a <code>port.packetFlowFileName</code> .

TABLE 5-87. stream Sub-Commands

Member	Usage
setQueue	For use with ATM ports only. Sets the parameters for a stream within a stream queue. See also atmPort , atmHeader , streamQueueList , and streamQueue .
getQueue	For use with ATM ports only. Gets the parameters associated with a stream in a queue.
export import	Write stream data to files and read it at a later time.
exportQueue importQueue	As in export/import but for a particular queue on ATM ports.
remove	Remove a stream from the port. See also stream remove chasID cardID portID streamID .

All of the stream sub-commands include an additional, optional argument named `sequenceType`. For POS cards that support DCC, this controls whether the stream is used by the DCC or normal data (SPE) channel.

streamRegion

The **streamRegion** command is used to manage several properties that apply to all streams. Refer to [streamRegion](#) on page A-780 for a full description of this

command. The important options of this class are mentioned in the following table:

TABLE 5-88. streamRegion Options

Member	Usage
gapControl	For ports that have the <i>portFeatureGapControlMode</i> capability, this controls the manner in which minimum inter-packet gaps are enforced.

weightedRandomFramesize

The **weightedRandomFramesize** command is used to configure possible different modes of generating random frame sizes for a particular stream. The following command is used for ports which support this feature and where the port has been programmed for random stream generation with:

```
stream config -frameSizeType sizeRandom
```

The following basic types of random streams are available:

- Uniform: Identical to previous implementations of the random framesize feature. A uniform set of random values between a minimum and maximum value are generated. The min/max values are set in the *stream* command's *frameSizeMIN* and *frameSizeMAX* options.
- Pre-programmed: A number of pre-programmed distributions are available, corresponding to standard traffic models found in various applications. See the *randomType* option description below.
- Custom: A distribution may be custom programmed for a stream. Pairs of frame size-weights are added to a list. Frame sizes may be any value valid for the port. Weights may be any value, such that the total of all of the weights is less than 2048. Pairs are added to the list using the *addPair* sub-command.
- Gaussian: Up to four gaussian curves may be summed up to generate a random distribution. Each curve is specified in the *center*, and *widthAtHalf* options and set by the *updateQuadGaussianCurve* sub-command. The *weight* option controls the distribution of values among the four curves.

For the pre-programmed and custom choices, the weights for all of the frame sizes are added up. Each frame size is then given a proportion of the total number of frames as dictated by its weight value. For example, one of the pre-programmed distributions is 64:7, 594:4, 1518:1. In this case, the total of the weights is 12 (7+4+1). Frames are randomly generated such that 64-byte frames are 7/12 of the total, 594-byte frames are 4/12 of the total and 1518-byte frames are 1/12 of the total.

Note that *stream* *get* must be called before this command's *get* sub-command. Refer to [weightedRandomFramesize](#) on page A-855 for a full description of this

command. The important options and sub-commands of this class are mentioned in the following table:

TABLE 5-89. weightedRandomFramesize Options

Member	Usage
randomType	The type of random distribution: uniform, pre-programmed, or custom.
pairList	Read-only list of framesize-weight pairs that are being used.
center	The values associated with one of the four Gaussian curves.
widthAtHalf	
weight	

TABLE 5-90. weightedRandomFramesize Sub-Commands

Member	Usage
addPair	Add a framesize-weight pair to the list.
delPair	Delete a pair from the list
updateQuadGaussianCurve	Sets the value for one of the four Gaussian curves.
retriveQuadGaussianCurve	Retrieves the values associated with one of the four Gaussian curves.

flexibleTimestamp

The **flexibleTimestamp** command is used to configure the placement of the time stamp. It normally is placed just at the end of the packet, just before the CRC. Time stamp insertion is controlled by the *enableTimestamp* option of the *stream* command.

The following basic types of time stamp placement are available:

- Before CRC: Just before the CRC at the end of the packet.
- At Offset: At a particular offset within the packet.

The important options of this class are mentioned in the following table:

TABLE 5-91. flexibleTimestamp Options

Member	Usage
type	The basic type of time stamp placement.
offset	If 'at offset' is used, then this is the offset to place the time stamp at.

Frame Data

udf

Up to five User Defined Fields may be defined, which allow arbitrary data to be algorithmically constructed within the data portion of the frames. The *get stream* sub-command must be called before *get udf* sub-command and *set*

`stream` sub-command must be called after `set udf` sub-command. See the *Ixia Reference Guide* for a general discussion. See [udf](#) on page A-822 for full details. The important options of this command are mentioned in the following table:

TABLE 5-92. **udf Options**

Member	Usage
enable	Enable or disable this UDF definition.
counterMode	Indicates the type of counter: <ul style="list-style-type: none"> • up-down • random • value list • nested • range list • IPv4.
offset	The offset within the packet to place the UDF data. Also known as Byte Offset.
countertype	The size and shape of the UDF counter. One to four 32 to 8 bit counters. Also known as Type. Note: On boards and modes that support <code>udfSize</code> , <code>countertype</code> is deprecated in favor of <code>udfSize</code> .
random	If set, all counters have random data.
continuousCount	If set, all counters operate continuously.
repeat	If <code>continuousCount</code> is not set, this is the repeat count for all counters.
updown	A four-bit mask indicating whether each counter counts up or down.
initval	The initial value of the counter.
maskselect/ maskval	Together these indicate which initial value bits to use and increment/decrement.
enableCascade	For PoS ports, enables a counter to continue with a count from stream to stream.
cascadeType	Indicates whether to cascade from the previous stream or the previous value on this stream.
step	The increment step for Up or Down increment mode can be specified. (Default = 1).
innerLoop innerRepeat innerStep	Controls the inner loop for a nested <code>counterMode</code> .
enableSkipZeros AndOnes skipMaskBits	Controls skipping over broadcast addresses for an IPv4 <code>counterMode</code> .
valueList	A list which holds the values to be used in value list mode.

TABLE 5-92. udf Options

Member	Usage
chainFrom	Allows you to select what UDF the current UDF should chain from. When this option is employed, the UDF stays in its initial value until the UDF it is chained from reaches its terminating value.
bitOffset	Origins from bit 0 (LSB) in the byte specified in <i>offset</i> (Byte Offset). The range is 0 to 7 bits.
udfSize	Sets the UDF field size (in bits). This must be a value from 1-32 and is only supported on certain cards in certain modes.

tableUdf

The *tableUdf* command is used to define tables of data that is applied at the same time as other UDFs. The *tableUdf* feature is only available for selected ports; the availability of the feature may be tested with the *port isValidFeature... portFeatureTableUDF* command.

The feature is enabled with the *enable* option. Tables consist of rows and columns. Columns define the locations within a packet that are to be modified, while rows hold the data that is simultaneously applied at the locations indicated by the columns. Columns are defined with *tableUdfColumn*; column attributes include the following:

- Column name
- Offset and size
- Data format; for example, IPv4 address.

Columns are then added to the table using the *addColumn* sub-command of this command. Once columns have been defined, data is added to the table, row by row, using the *addRow* sub-command.

Table UDF configurations, including row data, may be saved to disk using the *export* sub-command; a comma separated values (csv) file format is used. Table UDF configurations may be retrieved using the *import* sub-command.

The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-93. tableUdf Options

Member	Usage
enable	Enables the use of the table UDF.
numColumns	Read-only indicates of the current table size.
numRows	

TABLE 5-94. tableUdf Sub-Commands

Member	Usage
addColumn	Add a new column to the table from the data in the tableUdfColumn command.
addRow	Add a new row of data to the table.
clearColumns	Clear all columns and row data, access columns in the table.
getFirstColumn	
getNextColumn	
clearRows	Clear all row data, access rows in the table.
getFirstRow	
getNextRow	
export	Save and retrieve table UDF data to the file system.
import	

tableUdfColumn

The *tableUdfColumn* command is used columns used in table UDFs. Columns define the locations within a packet that are to be modified. Columns are defined with the options of this command and then added to a table using the *addColumn* sub-command of the [tableUdf](#) command. Column attributes include the following:

- Column name
- Offset and size: Data for multiple columns may not overlap
- Data format; for example, IPv4 address.

Column data for existing tables is retrieved with the *getFirstColumn* and *getNextColumn* sub-commands of the [tableUdf](#), the values retrieved are available in this command. The important options of this command are mentioned in the following table:

TABLE 5-95. tableUdfColumn Options

Member	Usage
name	Name of the column
offset	Offset in the packet to the start of the data.
size	Size of the data within the packet. Columns in a table UDF may not overlap.
formatType	
customFormat	The format of the data expected for the column. This is applied by the <i>addRow</i> sub-command of tableUdf .

tcpRoundTripFlows

The *tcpRoundTripFlows* command sets up values to be used in measuring round-trip times. See the *Ixia Reference Guide* for a general discussion. See [tcpRoundTripFlow](#) on page A-806 for full details.

The important options of this command are listed in the [Table 5-96](#) below.

TABLE 5-96. `tcpRoundTripFlows` options

Category	Member	Usage
Data	patternType	Dictates the type of data pattern manipulation: <ul style="list-style-type: none"> • increment/decrement bytes or words • random data • fixed repeating pattern chosen from <code>dataPattern</code> • fixed non-repeating pattern chosen from <code>dataPattern</code>
	dataPattern	One of a number of fixed patterns of data, including all 1's and all 0's, plus a choice for a user specified pattern.
	pattern	If <code>dataPattern</code> indicates a user specified pattern, this string specifies the contents.
Addresses	framesize	The number of bytes in each package.
	macSA	The source MAC address used for outbound packets.
	macDA	The destination MAC address used for outbound packets. This may be overridden through the use of <code>useArpTable</code> .
	useArpTable	If set, the ARP table is used instead to set the MAC address based on the destination IP address. <code>gatewayIpAddr</code> is used for the ARP query.
	gatewayIpAddr	If <code>useArpTable</code> is set, this is the address of the gateway that responds to ARP requests.
IP	forceIpSA	If set, IpSA is used to set the outbound IP address.
	IpSA	Outbound IP source address.

packetGroup

The `packetGroup` command sets up values to be used in measuring latency, classed by tagged groups of packets. See the *Ixia Reference Guide* for a description of this feature. See [packetGroup](#) on page A-469 for full details. To calculate latency values the `fir` object in the `stream` command should be set to `true` and the value of the port's `receiveMode` option should be set to `portPacketGroup`.

An additional feature available on some port types is the ability to collect latency measurements per packet group. The availability of this feature for a given port can be tested using the `port is ValidFeature... portFeatureRxLatencyBin`. The port must be configured for wide packet groups (the port's `receiveMode` includes the

portRxModeWidePacketGroup bit); the availability of this mode may be tested with *port* *isValidFeature... portFeatureRxWidePacketGroups*. No configuration is necessary on the transmit port; only the receive port must be configured to receive latency bin operation. This feature is enabled on the receive port with the *enableLatencyBins* option.

The latency measurements for each packet group may be collected in a set up to 16 continuous latency buckets. The first bucket always starts at 0 and the last bucket always ends at the maximum latency. The *packetGroup* interface allows for the specification of up to 15 time dividers between latency bins. For example, to specify five latency buckets for the following:

- 0 - 0.70ms
- 0.70ms - 0.72ms
- 0.72ms - 0.74ms
- 0.74ms - 0.76ms
- 0.76ms - max

one programs four dividing times:

- 0.70ms
- 0.72ms
- 0.74ms
- 0.76ms

This is done through the *latencyBins* option. No other setup is required for the receive side port. The latency statistics per latency bin are obtained through the use of the *packetGroupStats* command.

An additional feature available on some port types is the ability to measure latency as it varies over time. The availability of this feature for a given port can be tested using the *port* *isValidFeature... portFeatureRxTimeBin*. The port must be configured for wide packet groups (the *port's receiveMode* includes the *portRxModeWidePacketGroup* bit); the availability of this mode may be tested with *port* *isValidFeature... portFeatureRxWidePacketGroups*. No configuration is necessary on the transmit port; only the receive port must be configured to receive time bin operation. This feature is enabled on the receive port with the *enableTimeBins* option.

The latency over time for each packet group may be collected for a number of evenly spaced time periods, as indicated by the *numTimeBins* and *timeBinDuration* options. The number of packet groups used per time bin must also be specified in the *numPgidPerTimeBin* option.

The product of *numPgidPerTimeBin* (which must be a power of 2) and the next higher power of 2 of the *numTimeBins* must be less than the total number of packet group IDs available for the port when not in time bin mode.

The latency statistics per time bin are obtained through the use of the *packetGroupStats* command. Time bins and latency bins may be used at the same

time. The important options and sub-commands of this class are mentioned in the following table:

TABLE 5-97. packetGroup Options

Category	Member	Usage
Signature	signatureOffset	Where to place the signature in each transmitted packet.
	signatureMask	Masks specific bits for signature matching.
	signature	The signature to be inserted at the signature offset.
	insertSignature	Whether to insert the signature or not.
Packet ID	groupIdMode	Allows the group ID offset to be placed at one of several common locations, or set to a custom location indicated in <i>groupIdOffset</i> and <i>groupIdMask</i> .
	groupIdOffset	Where to place the group ID in each transmitted packet.
	enableGroupIdMask	Masks specific bits for group ID identification.
	groupIdMask	
	groupId	The value to use as the packet group ID.
Receive mode	maxRxGroupId	Displays the maximum number of PGIDs based on the port receive configuration.
	preambleSize	The expected size of the received preamble.
	latencyControl	The type of latency measurement are as follows: <ul style="list-style-type: none"> • <i>cutThrough</i>: First data bit in to first data bit out • <i>storeAndForward</i>: Last data bit in to first data bit out • <i>storeAndForwardPreamble</i>: Last data bit in to first preamble out • <i>interArrivalJitter</i>: The jitter between packet arrivals
	enableLatencyBins	Enables and sets the latency bins dividers.
	latencyBinList	
	enableTimeBins	Enables and sets up the time bin parameters: <ul style="list-style-type: none"> • Number of packet group IDs per time bin • Number of time bins • The size of each time bin
	enable128kBinMode	
	numPgIdPerTimeBin	
	numTimeBins	
	timeBinDuration	

TABLE 5-97. packetGroup Options

Category	Member	Usage
	enableFilterMask headerFilterMask	Controls filtering of incoming packets to a specific pattern. Only filtered packets are used for packet group matching.
Sequence checking	sequenceChecking-Mode	The mode of sequence checking: <ul style="list-style-type: none">• Based on out-of-sequence checking.• Based on multi-switched path loss and duplication checking.
	sequenceNumber-Offset	For out-of-sequence checking, where the sequence number is located.
	sequenceError-Threshold	For out-of-sequence checking, the threshold that is called an error.
	multiSwitchedPath-Mode	Controls the use of time stamps in multi-switched path mode.
Header	headerFilter	A value to be matched in the packet before signature matching occurs.
	headerFilter Mask	A value used to mask the headerFilter.

TABLE 5-98. packetGroup Sub-Commands

Member	Usage
setTx	Sets the packet group transmit characteristics for the port.
getTx	Gets the packet group transmit characteristics for the port.
setRx	Sets the packet group receive characteristics for the port.
getRx	Gets the packet group receive characteristics for the port.

dataIntegrity

The `dataIntegrity` class sets up values to be used to check data validity. See the *Ixia Reference Guide* for a description of this feature. In order for data integrity to operate, `port -receiveMode portRxDataIntegrity` must be performed (and committed to the hardware). See [dataIntegrity](#) on page A-117 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-99. dataIntegrity Options

Category	Member	Usage
Signature	signatureOffset	Where to place the signature in each transmitted packet.
	signature	The signature to be inserted at the signature offset.

TABLE 5-99. dataIntegrity Options

Category	Member	Usage
	insertSignature	Whether to insert the signature or not.
Receive mode	enableTimeStamp	Enables the placement of a 48-bit timestamp just before the FCS value with a 20ns accuracy.

TABLE 5-100. dataIntegrity Sub-Commands

Member	Usage
setTx	Sets the data integrity transmit characteristics for the port.
getTx	Gets the data integrity transmit characteristics for the port.
setRx	Sets the data integrity receive characteristics for the port.
getRx	Gets the data integrity receive characteristics for the port.

Sequence Checking

See the *Ixia Reference Guide* for a general discussion. There is no specific command that controls the operation of sequence checking. Instead, the following steps should be used to enable sequence checking:

1. Set the value of the port's `receiveMode` option to `portRxSequenceChecking`.
2. Set the location of the signature in the packet through the `signatureOffset` and `signature` members of the `packetGroup` command.
3. Set the location of the sequence check value through the `groupIdOffset` member of the `packetGroup` class.
4. Set the values to the hardware through the `setTx` sub-command of the `TclPacketGroup` command.
5. The statistics values `signatureErrors` and `signatureValues` relate to signature checking. Refer to [stat](#) on page A-703.

forcedCollisions

Collisions may be forced on the data transmission from any port. Refer to the *Ixia Reference Guide* for the full discussion of this feature and to [forcedCollisions](#) on page A-244 for command details. The important options and sub-commands of this class are mentioned in the following table:

TABLE 5-101. forcedCollisions options

Member	Usage
enable	Enables forced collisions.
packetOffset	The offset from the beginning of packet to the start of the collision.
collisionDuration	The duration of each collision

TABLE 5-101. forcedCollisions options

Member	Usage
consecutiveCollisions	The number of consecutive collisions to generate.
consecutiveNonColliding Packets	After each time that collisions have occurred, this is the number of packets that is not be modified.
continuous	Indicates that collisions are to occur continuously.
repeatCount	For non-continuous operation, the number of times to repeat the cycle of collisions and non-collisions.

Protocols

The following commands relate to protocol selection, header options and, in some cases, message formatting.

protocol

Basic protocol parameters are set with the `protocol` command. See [protocol](#) on page A-576 for full details. The important options of this command are mentioned in the following table:

TABLE 5-102. protocol Options

Category	Member	Usage
Protocol	appName	The protocol running on top of IP are the following: UDP, ARP, RIP, DHCP, DHCPv6, SRP, or PTP.
	name	Protocol selected: MAC, IP, IPv4, IPv6, IPX, pauseControl, and FCoE.
Data Link Layer	enable802dot1qTag	Enables 802.1q single or stacked VLAN tagged frames.
	enableISLtag	Enables Cisco ISL tagged frames.
	enableMPLS	Enables MPLS tagged frames.
	enableMacSec	Enable MacSec frame insertion in streams.
	enableOam	Enable OAM frame insertion in streams.
	ethernetType	Type of ethernet frame: EthernetII, IEEE802.3, IEEE802.3 Snap, or IEEE802.2

protocolOffset

The `protocolOffset` command allows the IP and other headers to be relocated from their default location to further into the packet. This allows additional headers to be inserted ahead of the protocol header, for example PPP headers. See [protocolOffset](#) on page A-580 for full details. The important options of this command are mentioned in the following table:

TABLE 5-103. protocolOffset Options

Member	Usage
offset	The new offset of the protocol header
userDefinedTag	The contents of the space where the protocol header was previously located.

fibreChannel

The *fibreChannel* command supports FCoE header and trailer in streams. See [fibreChannel](#) on page A-199 for details.

fcoe

The *fcoe* command is used to configure Fibre Channel over Ethernet (FCoE) header and trailer packet. FCoE is a method of communicating data for streams and protocols. See [fcoe](#) on page A-162 for details. Associated commands include [fcoeDiscoveredInfo](#) on page A-174, [fcoeProperties](#) on page A-179, [fibreChannel](#) on page A-199, and [npivProperties](#) on page A-427.

Data Link Layer

isl

The *isl* command sets up the header for Cisco ISL messages. The *enableISLtag* option of the *protocol* command must be set. The data portion of the message, including the tunneled SA and DA may be created using *stream*. See [isl](#) on page A-382 for full details.

The important options of this command are mentioned in the following table:

TABLE 5-104. *isl* Options

Member	Usage
isIDA	The multicast address indicating to the receiver that this is an ISL formatted packet.
frameType	The type of frame being encapsulated are as follows: <ul style="list-style-type: none">• Ethernet• Token Ring• FDDI• ATM
userPriority	Two bits of packet priority.
isISA	(Read-only) The source MAC address. The upper three bytes are reflected in the <i>hsa</i> field.
vlanID	The virtual LAN identifier.
bpdu	Set for all Bridge Protocol Data Units that are encapsulated by the ISL packet.
index	For diagnostic purposes, the port index of the packet as it exits the switch.
reserved	The ‘reserved’ field of the packet.

vlan

The *vlan* command sets up the header for VLAN specific messages. The *enable802dot1qtag* option of the *protocol* command must be set. The data

portion of the message may be created using `stream`. See [vlan](#) on page A-841 for full details. The important options of this command are mentioned in the following table:

TABLE 5-105. `vlan` Options

Member	Usage
<code>userPriority</code>	User priority level.
<code>cfi</code>	Canonical format indicator bit.
<code>vlanID</code>	The VLAN identifier.
<code>mode</code>	Indicates whether the VLAN tag is incremented/decremented or random between packets.
<code>repeat</code>	For each new value of the VLAN ID, this is the number of times it is repeated before the next change.
<code>step</code>	The step size between incremented/decremented values.
<code>maskval</code>	This masks the values of the VLAN ID that may be changed.

stackedVlan

The `stackedVlan` command is used to configure an ordered stack of VLAN entries. This command is only used when the `enable802dot1qTag` in the `protocol` command is set to `true`. Elements of the stack are constructed in the [vlan](#) command. The top two elements of the stack are always present and may be modified by using the `setVlan` sub-command. Other elements are added to the bottom of the stack using `addVlan`; they may later be modified with the `setVlan` sub-command.

The top two VLANs in a stack may be configured to increment or decrement their VLAN ID. They may either increment/decrement independently or operate in a special nested mode. To use nested mode, the top (outer) VLAN should be set to one of the non-nested increment/decrements modes and the second (inner) VLAN should be set to the nested increment or decrement mode. In this mode the inner VLAN's ID changes most rapidly.

See [stackedVlan](#) on page A-700 for full details. The important sub-commands and options of this command are mentioned in the following table:

TABLE 5-106. `stackedVlan` Sub-Commands

Member	Usage
<code>addVlan</code>	Adds a VLAN specification to the bottom of the stack.
<code>delVlan</code>	Deletes a specify VLAN from the list
<code>setVlan</code>	Changes the values of a VLAN stack element in place.
<code>getFirst/getNext</code>	Cycles through the stack of VLAN elements.

TABLE 5-107. stackedVlan Options

Member	Usage
numVlans	The read-only count of the number of defined VLANs.

mpls

The *enableMPLS* member of the associated *protocol* command must be set to successfully use this command. The *mpls* command sets up the base information for MPLS specific messages. The data portion of the message may be created using *mplsLabel*. See *mpls* on page A-416 for full details. Note that *stream get* must be called before this command's *get* sub-command. The important options of this command are mentioned in the following table:

TABLE 5-108. mpls Options

Member	Usage
type	The MPLS type: Unicast or multicast.
forceBottomOfStack	Automatically set bottom of stack bit.

mplsLabel

The *enableMPLS* member of the associated *protocol* object must be set to successfully use this command. The *mplsLabel* command is used to generate MPLS labels. See *mplsLabel* on page A-420 for full details. The important options of this command are mentioned in the following table:

TABLE 5-109. mpls Options

Member	Usage
label	The value of the label is one of the following: <ul style="list-style-type: none"> • IPV4 Explicit Null • Router Alert • IPV6 Explicit Null • Implicit Null • Reserved
experimentalUse	Sets the experimental use field.
bottomOfStack	Sets the bottom of stack bit.

IPX

The *ipx* command provides for the setting of IPX header elements. See *ipx* on page A-375 for full details. Note that *stream get* must be called before this

command's *get* sub-command. The important options of this command are mentioned in the following table:

TABLE 5-110. ipx options

Category	Member	Usage
Source	sourceNetwork	The network number of the source node.
	sourceNetworkCounterMode	Indicates whether the network number increments, decrements or receives a random setting.
	sourceNetworkRepeatCounter	The number of times that the network number changes.
	sourceNetworkMaskValue sourceNetworkMaskSelect	Together these set the mask of network number bits that changes.
	sourceNode sourceNodeCounterMode sourceNodeRepeatCounter sourceNodeMaskValue sourceNodeMaskSelect	As in sourceNetwork... for the sourceNode.
	sourceSocket sourceSocketCounterMode sourceSocketRepeatCounter sourceSocketMaskValue sourceSocketMaskSelect	As in sourceNetwork... for the sourceSocket.
Destination	destNetwork destNetworkCounterMode destNetworkRepeatCounter destNetworkMaskValue destNetworkMaskSelect	As in sourceNetwork... for the destNetwork.
	destNode destNodeCounterMode destNodeRepeatCounter destNodeMaskValue destNodeMaskSelect	As in sourceNetwork... for the destNetwork.
	destSocket destSocketCounterMode destSocketRepeatCounter destSocketMaskValue destSocketMaskSelect	As in sourceNetwork... for the destNetwork.
Misc	length	The total length of the IPX packet, including header. May be set if lengthOverride is set.
	lengthOverride	Allows the length value to be changed.

TABLE 5-110. ipx options

Category	Member	Usage
	packetType	The type of IPX packet.
	transportControl	The number of routers that the packet has passed through.

ARP

The *name* of the associated *protocol* object must be set to ‘ip’ and the *appName* must be set to ‘arp’ to successfully use this command. The arp command allows ARP packets to be constructed. See [arp](#) on page A-2 for full details.

The important options of this command are mentioned in the following table:

TABLE 5-111. arp options

Category	Member	Usage
Basic	operation	The type of ARP operation is one of the following: <ul style="list-style-type: none"> • ARP Request • ARP Reply • Reverse ARP Request • Reverse ARP Reply
Physical	hardwareAddrLength	(Read-only) The length of the hardware address.
	hardwareType	(Read-only) The hardware type of the physical layer.
	protocolAddrLength	(Read-only) The length of the protocol addresses.
	protocolType	(Read-only) The type of network protocol address.
Source	sourceHardwareAddr	The MAC address of the sender.
	sourceProtocolAddr	The protocol address of the sender.
Destination	destHardwareAddr	The MAC address of the receiver.
	destProtocolAddr	The protocol address of the receiver.

IP

The `ip` command allows IPv4 header values to be constructed. The name of the associated `protocol` object must be set to ‘ip’ to successfully use this command.

The source and destination addresses may be set from the result of a PPP negotiation through the use of the `enableDestSyncFromPpp` and `enableSourceSyncFromPpp` options. Note that it is necessary to wait until the PPP session has been negotiated before doing the following:

- performing a *chassis refresh* command
- performing a *stream get* command
- performing an *ip get* command
- reading the `destAddr` and `sourceAddr` values using `ip cget`

See [ip](#) on page A-320 for full details. The important options of this command are mentioned in the following table:

TABLE 5-112. ip Options

Category	Member	Usage
Header	precedence delay throughput reliability cost reserved	The parts of the Type of Service (TOS) byte.
	qosMode dscpMode dscpValue classSelector assuredForwardingClass assuredForwardingPrecedence	Allows the specification of the TOS byte using DSCP (DiffSrv).
	totalLength	The total length of the IP packet, including header. This may be overridden from the automatically calculated setting if <code>lengthOverride</code> is set.
	lengthOverride	Allows the <code>totalLength</code> of the packet to be overridden from the calculated setting.
	identifier	An identifier used to re-assemble fragments.
	fragment	Indicates whether this is a fragmented datagram.

TABLE 5-112. ip Options

Category	Member	Usage
	fragmentOffset	For fragmented packets, the offset in the re-assembled datagram where this packet's data belongs.
	lastFragment	Indicates that this is the last fragment of the datagram.
	ttl	Time to live for packet, in seconds.
	ipProtocol	The next level protocol contained in the data portion of the packet.
	useValidChecksum	Indicates whether a valid or invalid checksum should be included in the header.
	checksum	(Read-only) The value of the header checksum. Only valid after <code>stream.set</code> is performed.
Source	sourcelpAddr	The source IP address.
	sourcelpAddrMode	Indicates how the IP address changes from packet to packet: Remains the same, increments/decrements host, network, or random values.
	sourcelpAddrRepeatCount	The number of different source addresses generated.
	sourcelpMask	The source IP subnet mask.
	sourceCommand	The command type for the source address: A, B, C, D, or no command.
	enableSourceSyncFromPpp	Sets the <code>sourcelpAddr</code> from the result of a PPP negotiation.
Destination	destIpAddr	The destination IP address.
	destIpAddrMode	Indicates how the IP address changes from packet to packet: Remains the same, increments/decrements host, network, or random values.
	destIpAddrRepeatCount	The number of different destination addresses generated.

TABLE 5-112. ip Options

Category	Member	Usage
	destIpMask	The destination IP subnet mask.
	destClass	The class type for the destination address: A, B, C, D, or no class.
	enableDestSyncFromPpp	Sets the <i>destIpAddr</i> from the result of a PPP negotiation.
Misc	options	Variable length options field.
	destDutIpAddr	The address of the device under test.
	destMacAddr	The MAC address of the device under test. Received ARP frames modifies this value.

ipV6

The *ipV6* command allows IPv6 header values to be constructed. The name of the associated *protocol* object must be set to ‘ipV6’ to successfully use this command. *ipV6Address* on page A-347 is a convenience command which can be used to encode/decode and IPv6 address. See *ipV6* on page A-335 for full details.

The *ipV6* command also includes list of extension headers. Extension headers are built-in with the following specific objects:

- *ipV6Authentication*
- *ipV6Destination*
- *ipV6Fragment*
- *ipV6HopByHop*
- *ipV6Routing*

An extension header is added to a *ipV6* object by configuring the extension header with the appropriate command from the list above and then adding it to the group with *ipV6 addExtensionHeader type*, where *type* indicates which of the extensions to use. An extension may be retrieved from an *ipV6* object through the use of *getFirstExtensionHeader / getNextExtensionHeader*. These commands return the **name** of the command that was used to configure the header extension. The type of the extension header can be determined from the *nextHeader* value from the *ipV6* command (for the first extension header) or from the **previous** extension header otherwise. This is typically used in the following sequence of commands:

```
set eHeader [ipV6 getFirstExtensionHeader]
set nextType [$eHeader cget -nextHeader]
```

In addition, if this is to be the header to a TCP, UDP or ICMP packet, then a separate call to *ipV6 addExtensionHeader* must be made with *tcp*, *udp* or *icmpV6* must be made. For example:

```
ipV6 addExtensionHeader tcp
```

Although it is the default, *ipV6 addExtensionHeader ipV6NoNextHeader* may be used to indicate that there is no header following this one.

Note that *stream get* must be called before this command's *get* sub-command.

The source and destination addresses may be set from the result of a PPP negotiation through the use of the *enableDestSyncFromPpp* and *enableSourceSyncFromPpp* options. Note that it is necessary to wait until the PPP session has been negotiated before doing the following:

- performing a *chassis refresh* command
- performing a *stream get* command
- performing an *ipv6 get* command
- reading the *destAddr* and *sourceAddr* values using *ipv6 cget*

The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-113. ipV6 Options

Category	Member	Usage
Header	trafficClass	The traffic class of the IPv6 header.
	flowLabel	The flow label of the IPv6 header.
	hopLimit	The hop limit of the IPv6 header.
	sourceAddr	The source IPv6 address.
	sourceMask	The mask associated with the source address.
	sourceAddrMode	The manner in which to modify the source address between iterations: idle or increment/decrement network/host as well as address prefix dependent sub-components of the address.
Source	sourceAddrRepeatCount	The number of times to modify the address before restoring it to <i>sourceAddr</i> .
	sourceAddrStepSize	The size of the increment for increment/decrement modes.
	enableSourceSyncFromPpp	Sets the <i>sourceAddr</i> from the result of a PPP negotiation.
Destination	destAddr	The destination IPv6 address.

TABLE 5-113. ipV6 Options

Category	Member	Usage
Misc	destMask	The mask associated with the destination address.
	destAddrMode	The manner in which to modify the destination address between iterations: idle or increment/decrement network/host.
	destAddrRepeatCount	The number of times to modify the address before restoring it to <i>destAddr</i> .
	destAddrStepSize	The size of the increment for increment/decrement modes.
	enableDestSyncFromPpp	Sets the <i>destAddr</i> from the result of a PPP negotiation.
	payloadLength	(Read-only) The calculated length of the packet payload
	nextHeader	(Read-only) The type of the next payload header

TABLE 5-114. ipV6 Sub-Commands

Member	Usage
clearAllExtensionHeaders	Removes all extension headers.
addExtensionHeader	Adds a new extension header.
getFirstExtensionHeader	Iterates through the extension headers.
getNextExtensionHeader	

ipV6Authentication

The *ipV6Authentication* command is used to build an IPv6 authentication header to be added to an *ipV6* header using *ipV6 addExtensionHeader*. See *ipV6Authentication* on page A-352 for full details. The important options of this command are mentioned in the following table:

TABLE 5-115. ipV6Authentication options

Member	Usage
nextHeader	(Read-only) The type of the next header in the IPv6 header.
authentication	A variable length string containing the packets integrity check value (ICV).
payloadLength	The length of the authentication data, expressed in 32-bit words.

TABLE 5-115. ipV6Authentication options

Member	Usage
securityParamIndex	The security parameter index (SPI) associated with the authentication header.
sequenceNumberField	A sequence counter for the authentication header.

ipV6Destination

The *ipV6Destination* command is used to build an IPv6 destination header to be added to an *ipV6* header using *ipV6 addExtensionHeader*. See *ipV6Destination* on page A-354 for full details. The important options of this command are mentioned in the following table:

TABLE 5-116. ipV6Destination options

Member	Usage
nextHeader	(Read-only) The type of the next header in the IPv6 header.

ipV6Fragment

The *ipV6Fragment* command is used to build an IPv6 fragment header to be added to an *ipV6* header using *ipV6 addExtensionHeader*. See *ipV6Fragment* on page A-356 for full details. The important options of this command are mentioned in the following table:

TABLE 5-117. ipV6Fragment options

Member	Usage
nextHeader	(Read-only) The type of the next header in the IPv6 header.
enableFlag	Indicates whether there are more fragments to be received (<i>true</i>) or this is the last fragment (<i>false</i>).
fragmentOffset	A 13-bit value which is the offset for the data contained in this packet, relative to the start of the fragmentable part of the original packet, in 8-octet units.
identification	A 32-bit value that uniquely identifies the original packet which is to be fragmented.

ipV6HopByHop

The *ipV6HopByHop* command is used to build an IPv6 Hop by Hop header to be added to an *ipV6* header using *ipV6 addExtensionHeader*. See *ipV6HopByHop* on page A-358 for full details.

The important options of this command are mentioned in the following table:

TABLE 5-118. *ipV6HopByHop* options

Member	Usage
getFirstOption	<i>Read-only.</i> Gets the first option configured in the packet.
getNextOption	<i>Read-only.</i> The type of the next option.
optionsList	Displays a list of the Hop by Hop headers included in the packet.

It is necessary to configure the Hop by Hop options before using this command, using the following Hop by Hop option commands:

- *ipV6OptionPAD1* on page A-361
- *ipV6OptionPADN* on page A-362
- *ipV6OptionJumbo* on page A-363
- *ipV6OptionRouterAlert* on page A-364
- *ipV6OptionBindingUpdate* on page A-365
- *ipV6OptionBindingAck* on page A-367
- *ipV6OptionHomeAddress* on page A-368
- *ipV6OptionBindingRequest* on page A-369
- *ipV6OptionMIPv6UniqueIdSub* on page A-370
- *ipV6OptionMIPv6AlternativeCoaSub* on page A-371
- *ipV6OptionUserDefine* on page A-372

ipV6Routing

The *ipV6Routing* command is used to build an IPv6 routing header to be added to an *ipV6* header using *ipV6 addExtensionHeader*. See *ipV6Routing* on page A-373 for full details. The important options of this command are mentioned in the following table:

TABLE 5-119. *ipV6Routing* options

Member	Usage
nextHeader	(<i>Read-only</i>) The type of the next header in the IPv6 header.
nodeList	A list of 128-bit IPv6 addresses, which may be constructed with the <i>ipV6Address</i> command.

tcp

The name of the associated `protocol` object must be ‘ip’ and the `ipProtocol` member of the associated `ip` object must be set to ‘tcp’ to successfully use this command. The `tcp` command allows TCP header values to be constructed. See [tcp](#) on page A-802 for full details. Note that `stream get` must be called before this command’s `get` sub-command. The important options of this command are mentioned in the following table:

TABLE 5-120. **tcp** options

Category	Member	Usage
Header	offset	Offset from the beginning of the header to the packet data.
	sourcePort	Source port number.
	destPort	Destination port number.
	sequenceNumber	Packet sequence number.
	acknowledgement Number	Next byte that the receiver expects.
	window	The number of bytes the recipient may send back, starting with the ack byte.
	urgentPointer	Byte offset to urgent data with the packet.
	checksum	Read-only checksum after a <code>decode</code> operation.
Flags	urgentPointerValid	Indicates whether the <code>urgentPointer</code> field is valid.
	acknowledgeValid	Indicates whether the <code>acknowledgementNumber</code> is valid.
	pushFunctionValid	Request that the receiver push the packet to the receiving application without buffering.
	resetConnection	Resets the connection.
	synchronize	Indicates either a connection request or acceptance.
	finished	Indicates that this is the last packet to be sent for the connection.
	useValidChecksum	Indicates whether a valid or specified checksum is to be used.

udp

The name of the associated `protocol` object must be set to ‘ip’ and the `appName` member must be set to ‘udp’ to successfully use this command. The `udp` command is used to format UDP headers. See [udp](#) on page A-831 for full details. Note that `stream get` must be called before this command’s `get` sub-command. The important options of this command are mentioned in the following table:

TABLE 5-121. udp Options

Member	Usage
<code>sourcePort</code>	Port of the sending process.
<code>destPort</code>	Port of the destination process.
<code>length</code>	Length of the datagram including the header. If <code>lengthOverride</code> is set, this value is used instead of the calculated value.
<code>lengthOverride</code>	Allows the length parameter to be used to set the packet length.
<code>enableChecksum</code>	Causes a valid or invalid checksum to be generated in the UDP header.
<code>checksum</code>	The actual value generated. Valid only after <code>stream.set</code> has been used.
<code>enableChecksumOverride</code>	Enables the setting of a checksum from <code>checksum</code> .
<code>checksumMode</code>	Indicates whether to use the correct checksum or its invalid one’s complement.

igmp

The name of the associated `protocol` object must be set to ‘ip’ and the `ipProtocol` member of the associated `ip` object must be set to ‘igmp’ to successfully use this command. The `igmp` command is used to format IGMP messages. See [igmp](#) on page A-292 for full details. Note that `stream get` must be called before this command’s `get` sub-command. The options and sub-commands are mentioned in the following table:

TABLE 5-122. igmp Options

Member	Usage
<code>version</code>	Which version of IGMP to use: 1, 2, or 3.
<code>type</code>	The type of IGMP message to generate: <ul style="list-style-type: none"> • membership Query • membership Report: type 1, 2, or 3. • DVMRP Message: Distance Vector Multicast Routing Protocol • Leave Group
<code>maxResponseTime</code>	The maximum allowed response time.

TABLE 5-122. igmp Options

Member	Usage
groupIpAddress	The IP multicast address of the group being joined or left.
mode	Describes how groupIpAddress changes from one message to the next: idle, increment or decrement.
repeatCount	The number of IGMP messages to send.
useValidChecksum	Use a valid or over-written checksum.
qqic	Options used for an IGMP v.3 group membership request.
qrv	
enableS	
sourceIpAddressList	

TABLE 5-123. igmp Sub-Commands

Member	Usage
clearGroupRecords	Clears the group records list.
addGroupRecord	Adds a group record from the <i>igmpGroupRecord</i> command to the list.
getFirstGroupRecord	Iterates through the group records or accesses one directly.
getNextGroupRecord	
getGroupRecord	

igmpGroupRecord

The *igmpGroupRecord* command holds a Group Record to be included in an IGMPv.3 group membership report. See *igmpGroupRecord* on page A-297 for full details. The important options of this command are mentioned in the following table:

TABLE 5-124. igmpGroupRecord Options

Member	Usage
type	The type of the group record.
multicastAddress	The multicast address that this group record pertains to.
sourceIpAddressList	A set of source IP addresses for the multicast group.

icmp

The name of the associated *protocol* object must be set to ‘ip’ and the *ipProtocol* member of the associated *ip* object must be set to ‘icmp’ to successfully use this command. The *icmp* command is used to format ICMP messages. Any data not covered in the options below must be entered in the stream’s data portion. See *icmp* on page A-267 for full details. Note that *stream*

get must be called before this command's *get* sub-command. The important options of this command are mentioned in the following table:

TABLE 5-125. icmp Options

Member	Usage
type	The type of ICMP message to be sent.
code	The code for each type of message.
checksum	(Read-only) The value of the checksum to be sent in the stream. This is only valid after <code>stream.set</code> is used.
id	ID for each <code>echoRequest</code> type message.
sequence	Sequence number for each <code>echoRequest</code> type message.

rip

Note that the `rip` and `ripRoute` commands allow you to create RIP packets for transmission as part of a stream. They are not associated with the RIP aspect of the Protocol Server, described in [rip](#) on page A-608. The name of the associated `protocol` object must be set to 'ip' and the `appName` member of the associated `protocol` object must be set to 'Rip' to successfully use this command. The `rip` command is used to configure the RIP header. Individual RIP route entries are associated with the `ripRoute` command and the use of `RouteIds`. See [rip](#) on page A-608 for full details. Note that `stream.get` must be called before this command's *get* sub-command. The important options of this command are mentioned in the following table:

TABLE 5-126. rip Options

Member	Usage
command	The RIP command. One of the following options: <ul style="list-style-type: none"> • RIP Request • RIP Response • RIP Trace On/Off • RIP Reserved
version	The RIP version: 1 or 2.

ripRoute

The `ripRoute` command is used to format RIP route messages. Header information is contained in the associated `rip` command. See [ripRoute](#) on page A-612 for full details. The important options of this command are mentioned in the following table:

TABLE 5-127. ripRoute Options

Member	Usage
familyId	Address family identifier.
routeTag	Used to distinguish multiple sources of routing information.
ipAddress	IP address of the entry.

TABLE 5-127. ripRoute Options

Member	Usage
subnetMask	Subnet mask for the entry.
nextHop	For version 2 records only, the IP address of the next routing hop for the entry.
metric	The cost of the route, from 1 to 16.
authenticationType	The type of authentication to use.
authentication	Data associated with the authentication method.

dhcp

The name of the associated `protocol` object must be set to ‘ip’ and the `appName` member of the associated `protocol` object must be set to ‘Dhcp’ to successfully use this command. The `dhcp` command is used to format DHCP messages. Multiple options are entered into the message through repeated use of the `setOption` method.

Both DHCPv4 and DHCPv6 are supported.

A DHCP server may also be used to obtain an address for a port. Refer to [Using DHCP with Interfaces](#) on page 5-115 for full details.

See [dhcp](#) on page A-125 for full details. Note that `stream get` must be called before this command’s `get` sub-command. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-128. dhcp Options

Member	Usage
opCode	Operation code: <ul style="list-style-type: none">• DHCP Boot Request• DHCP Boot Reply
hwType	Hardware address type.
hwLen	Hardware address length.
hops	Set to 0 to indicate packet origin.
transactionID	Random identifier for message pairing.
seconds	Elapsed time since start of request.
flags	Indicates broadcast or non-broadcast handling.
clientIpAddr	Client’s IP address.
yourIpAddr	Your IP address.
relayAgentIpAddr	Relay agent’s IP address; used if booting through a proxy.
clientHWAddr	Client’s hardware address.
serverHostName	Optional server host name.

TABLE 5-128. dhcp Options

Member	Usage
serverIpAddr	Server's IP address.
bootFileName	Boot file name to use.
optionCode	Code for optional data.
optionDataLength	Length of the option data.
optionData	The actual data.

TABLE 5-129. dhcp Sub-Commands

Member	Usage
setOption	Sets an option value.
getOption	Gets a previously set option.
getFirstOption	Gets options by iterating through the list.
getNextOption	

ptp

Precision Time Protocol (PTP) enables precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing, and distributed objects. See [ptp](#) on page A-586 for details. Associated commands include [ptpAnnounce](#) on page A-592, [ptpDelayRequest](#) on page A-595, [ptpDelayResponse](#) on page A-596, [ptpDiscoveredInfo](#) on page A-597, [ptpFollowUp](#) on page A-599, [ptpProperties](#) on page A-600, and [ptpSync](#) on page A-603.

pauseControl

The `pauseControl` command is used to format pause control messages. The important options of this command are mentioned in the following table:

TABLE 5-130. pauseControl Options

Member	Usage
da	The MAC address of the interface receiving the pause control message. 10GE cards may set this value.
pauseTime	The pause time, measured in pause quanta units. (1 Pause Quanta = 512 bit times.) The valid range is 0 to 65535 pause quanta.
pauseControlType	Configure the priority control type, ieee 8023x or ieee 8021Qbb.
pauseFrame	Configure the hex byte priorities; 16 byte hex list.
pfcEnableValueList	Use to configure the priority parameters using pair list, where each pair contains the enable/disable value and pause quanta value.

srpArp

The `srpArp` command is used to format SRP based ARP messages for SONET cards. The SRP header options which are common to all of the SRP commands are mentioned in the following table:

TABLE 5-131. SRP Header Options

Member	Usage
mode	The type of packet: ATM cell, control message, usage message, or data packet.
parity	The ability to insert a good or bad parity bit in the header.
priority	The priority of the packet, from 0 through 7.
ringIdentifier	Whether the packet is intended for the inner or outer loop.
ttl	The time-to-live value.

The important additional options for the `srpArp` command are mentioned in the following table:

TABLE 5-132. srpArp Options

Member	Usage
macDestAddress	The destination MAC address for the ARP packet.
sourceDestAddress	The source MAC address for the ARP packet.

srpIps

The `srpIps` command is used to format SRP Intelligent Protection Switching (IPS) messages for SONET cards. The SRP header options which are common to all of the SRP commands are shown in Table 5-131 on page 5-85.

The additional options common to control messages (`srpIps` and `srpDiscovery`) are mentioned in the following table:

TABLE 5-133. SRP Control Message Options

Member	Usage
controlChecksum	Whether to insert a good or bad checksum.
controlTtl	The hop-count associated with the control message.
controlVersion	The protocol version number. Only 0 is currently supported.

The important additional options for the `srpIps` command are mentioned in the following table:

TABLE 5-134. srpIps Options

Member	Usage
originatorMacAddress	The original packet author's MAC address
pathIndicator	Whether the IPS packet is sent to the adjacent node or around the entire ring.
requestType	The type of IPS request.
statusCode	The IPS state of the sending node.

srpDiscovery

The `srpDiscovery` command is used to format SRP Discovery messages for SONET cards. The SRP header options which are common to all of the SRP commands are shown in Table 5-131 on page 5-85. The additional options common to control messages are shown in Table 5-133 on page 5-85.

The important additional options and sub-commands for the `srpDiscovery` command are mentioned in the following table:

TABLE 5-135. srpDiscovery Options

Member	Usage
topologyLength	The length of the topology data which follows.

TABLE 5-136. srpDiscovery Sub-Commands

Member	Usage
clearAllMacBindings	Clears the MAC bindings associated with the discovery packet.
addMacBinding	Adds a MAC binding from the <code>srpMacBinding</code> command to the list.
getFirstMacBinding	Iterates through the MAC bindings or addresses one directly.
getNextMacBinding	
getMacBinding	

srpMacBinding

The `srpMacBinding` command is used to format MAC bindings that are part of an SRP Discovery packets for SONET cards. The important additional options for the `srpDiscovery` command are mentioned in the following table:

TABLE 5-137. srpMacBinding Options

Member	Usage
address	The bound MAC address.

TABLE 5-137. srpMacBinding Options

Member	Usage
ringIdentifier	The ring to which the binding applies.
wrappedNode	Whether the node is currently wrapped or not.

srpUsage

The `srpUsage` command is used to format SRP Usage messages for SONET cards. The SRP header options which are common to all of the SRP commands are shown in Table 5-131 on page 5-85. The important additional options for the `srpUsage` command are mentioned in the following table:

TABLE 5-138. srpUsage Options

Member	Usage
rxMacAddress	The source MAC address for the usage packet.
rxTimeout	The receive timeout value.
rxTimeoutThreshold	The number of timeouts to wait before declaring the neighbor node as down.
txMacAddress	The destination MAC address of the usage packet.
txRepeatInterval	The interval at which usage packets are transmitted.
txUsageEnabled	Enables the repeated transmission of usage packets.
txValue	The data value to accompany the usage packet.

streamQueueList

See the *Ixia Reference Guide* for a general discussion. See [streamQueueList](#) on page A-778 for full details. ATM streams are organized into up to 15 queues, each queue may contain a number of streams. Up to 4096 streams may be distributed across the 15 queues. All queues are transmitted in parallel. The `streamQueueList` command adds and deletes stream queues to a port. Stream queues may also be automatically created with the `stream setQueue` command. The important options and sub-commands of this class are mentioned in the following table:

TABLE 5-139. streamQueueList Options

Member	Usage
avgDataBitRate	The average bit rate across all queues (read-only).
avgCellRate	The average cell rate across all queues (read-only).
avgPercentLoad	The average percentage load across all queues (read-only).
avgFramerate	The average framerate across all queues (read-only).

TABLE 5-140. streamQueueList Sub-Commands

Member	Usage
select	Selects the port to operate on.
clear	Removes all stream queues from a port.
add	Adds a stream queue to the port.
del	Deletes a stream queue from the port.

streamQueue

See [streamQueue](#) on page A-775 for full details. The *streamQueue* command sets the transmission rate for all of the streams in a queue. The important options and sub-commands of this class are mentioned in the following table:

TABLE 5-141. streamQueue Options

Member	Usage
rateMode	Control for and different means by which the ATM bit rate may be configured.
percentMaxRate	
aal5PduBitRate	
cellBitRate	
aal5FrameRate	Read-only. The ATM bit rate expressed in alternate units.
aal5PayloadBitRate	
aal5SduBitRate	
cellRate	
enableInterleave	Controls whether a stream queue's cells may be interleaved with other stream queues.

TABLE 5-142. streamQueue Sub-Commands

Member	Usage
clear	Removes all streams from a queue.

npiV

The **npiVProperties** command is used to configure an unconnected NPIV interface. (NPIV means N_Port_ID Virtualization). See [npiVProperties](#) on page A-427 for details.

Data Capture and Statistics

Data is captured as a result of the use of the following commands:

- *filter*: Sets up conditions under which data capture is triggered and filtered. *filter* sets up the conditions for collecting several user defined statistics.
- *filterPalette*: Sets up address and pattern matches used in *filter*.
- *capture*: Sets up basic sizing parameters for captured data.
- *captureBuffer*: Provides access to the raw data and latency/jitter measurements.
- *qos*: Sets up conditions under which QoS statistics are gathered.
- *atmReassembly*: Registers particular ATM VPI/VCIs for stream reassembly.
- *atmFilter*: Sets up ATM data and mask conditions and allows ATM data matches to be used for user defined statistics or capture trigger and filter.

Raw data and statistics are collected through the use of the following commands:

- *stat*: Provides access to all of the port statistics.
- *statGroup*, *statList* and *statWatch*: Provides access to average latency data and timestamps during packet group operation.
- *packetGroupStats*: Provides access to statistics organized by groups of ports.
 - *latencyBin*: Holds latency bin information.
- *vsrStat*: For 10Gigabit Ethernet VSR boards, provides access to global and per channel statistics.
- *vsrError*: For 10Gigabit Ethernet VSR boards, provides for the insertion of VSR errors.
- *atmStat*: For ATM boards, provides access to per VPI/VCI statistics.
- *streamTransmitStats*: For certain types of board, per-stream transmit statistics.

See the *Ixia Reference Guide* and the *Ixia Reference Guide* for a general discussion.

filter

filter sets up the conditions under which data capture is triggered and filtered. Conditions for the collection of user defined statistics (UDS) 1, 2, 5 and 6 are also specified. User defined statistics 5 and 6 are also known as async trigger 1 and 2. See *filter* on page A-218 for full details.

There are six sets of eight options for the capture trigger and filter and the four user UDFs. The following contribute a prefix to the option name:

- *captureTrigger...*
- *captureFilter...*

- userDefinedStat1...
- userDefinedStat2...
- asyncTrigger1...
- asyncTrigger2...

The options for the suffix to these names are mentioned in the following table:

TABLE 5-143. filter Options

Member	Usage
Enable	Enables or disables the filter, trigger or statistic.
DA	Two destination address matches (DA1 and DA2) are set through the use of <code>filterPallette</code> . This member chooses which conditions relating to those addresses are required for a match: <ul style="list-style-type: none"> • Any address • DA1 • Not DA1 • DA2 • Not DA2
SA	Two source address matches (SA1 and SA2) are set up through the use of <code>filterPallette</code> . This member chooses which conditions relating to those addresses are required for a match: <ul style="list-style-type: none"> • Any address • SA1 • Not SA1 • SA2 • Not SA2
Pattern	Two pattern matches (pattern1 and pattern2) are set up through the use of <code>filterPallette</code> . This member chooses which conditions relating to those pattern matches are required for a match: <ul style="list-style-type: none"> • Any address • pattern1 • Not pattern1 • pattern2 • Not pattern2 • pattern1 and pattern2
Error	The error condition under which a match occurs including the following: <ul style="list-style-type: none"> • Anytime, without concern over errors. • Only for good packets • Any of a number of other error conditions.
FrameSizeEnable	Enables or disables the size constraint as specified in the two entries below.
FrameSizeFrom FrameSizeTo	The minimum and maximum frame size for a match.

For example, at a minimum the *Enable* option of the *captureTrigger* command and the *Enable* option of the *captureFilter* command must be set for any data to be captured.

filterPallette

`filterPallette` sets up address and data pattern matching criteria used in `filter`. See [filterPallette](#) on page A-228 for full details.

There are four sets of two options for the source and destination addresses 1 and 2. These are mentioned in the following table:

TABLE 5-144. **filterPallette Options - DA/SA**

Member	Usage
DA1	Destination address 1 data.
DAMask1	Mask of valid bits for destination address 1.
DA2 / DAMask2	Same for destination address 2.
SA1 / SAMask1	Same for source address 1.
SA2 / SAMask2	Same for source address 2.

There are two sets of four options for each of the two data patterns. These are mentioned in the following table:

TABLE 5-145. **filterPallette Options - Pattern 1/2**

Member	Usage
matchType1	The basic form of match performed. This is one of a number of pre-programmed choices in which the packet type and data pattern are pre-programmed and/or specially interpreted. One additional choice allows for user specification of the data and type.
patternOffset1 patternOffsetType1	If the user choice is made in <code>matchType1</code> , this is the offset of pattern 1 in the frame. For some port types, it is possible to specify where the offset is with respect to; for example, from the start of the IP header.
pattern1	The data within the pattern to match for. For the pre-programmed choices in <code>matchType1</code> , this pattern has a special interpretation.
patternMask1	The mask to apply against <code>pattern1</code> to obtain a match.
patternOffset2 matchType2 pattern2 patternMask2	The same as for pattern 1, but for pattern 2.

In addition the following options control matching on GFP errors:

TABLE 5-146. filterPallette Options

Member	Usage
enableGfpBadFcsError enableGfpeHecError enableGfpPayloadCrc enableGfptHecError	Enables or disables the use of a particular GFP error condition.
gfpErrorCondition	Indicates whether the above enables need to all be present (AND'd) or just one (OR'd).

capture

`capture` sets up the basic parameters associated with the capture buffer usage. The capture process itself is started through the use of the *portGroup setCommand startCapture* command, or the *ixStartCapture* high-level command. The capture is stopped with the use of the *portGroup setCommand stopCapture* command, or the *ixStopCapture* high-level command, or a *captureBuffer get* command. That is, the act of reading the capture buffer stops the capture process. The high-level command, *ixCheckTransmitDone*, may be used to wait until all ports have finished transmitting.

See *capture* on page A-67 for full details. The important options of this command are mentioned in the following table:

TABLE 5-147. capture Options

Member	Usage
<code>sliceOffset</code>	The offset within the frame from which to begin capturing data.
<code>sliceSize</code>	The maximum number of octets per frame to capture. 8192 is the largest slice size supported.
<code>nPackets</code>	(Read-only) The actual number of packets available in the capture buffer.

captureBuffer

`captureBuffer` allows the raw captured data to be obtained, or calculated latency data to be viewed. Data is held in the hardware until the `get` method is called, which copies the captured data for a range of frame numbers into local computer memory. Following the use of `get`, `getframe` makes an individual frame available. Latency and deviation values may be calculated, subject to constraints through the use of `setConstraint` and `getStatistics`. Latency is defined as the difference between the transmit and receive times, in nanoseconds. Jitter is defined as the deviation of the latency. See *captureBuffer* on page A-73 for full details.

The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-148. captureBuffer Options

Category	Member	Usage
Data	frame	(Read-only) The contents of the selected frame based on <code>sliceSize</code> .
	length	(Read-only) The total length of the frame, regardless of the slice captured.
	numFrames	The number of frames in the hardware's capture buffer. After <code>setConstraints</code> is called, this value is updated with the number of frames that met the constraints.
	status	The status of the frame: either no errors, or one of a number of possible error conditions.
	timestamp	The arrival time of the captured frame in nanoseconds.
Measurements	averageLatency	(Read-only) The average latency of the frames in the retrieved capture buffer (in nanoseconds).
	latency	(Read-only) The frame's latency (in nanoseconds).
	minLatency	(Read-only) The minimum latency (in nanoseconds) of the frames in the retrieved capture buffer.
	maxLatency	(Read-only) The maximum latency (in nanoseconds) of the frames in the retrieved capture buffer.
	averageDeviation	(Read-only) The average deviation of the average latencies of the frames in the retrieved capture buffer.
Constraints	standardDeviation	(Read-only) The standard deviation of the average latencies of the frames in the retrieved capture buffer.
	enableEthernetType	Enables jitter calculations to occur only over those frames with the ethernet type indicated in <code>ethernetType</code> .
	ethernetType	If <code>enableEthernetType</code> is set, this is the ethernet type to match on.
	enableFramesize	Enables jitter calculations to occur only over those frames with the frame size indicated in <code>framesize</code> .

TABLE 5-148. captureBuffer Options

Category	Member	Usage
	framesize	If enableFramesize is set, this is the frame size to match on.
	enablePattern	Enables jitter calculations to occur only over those frames with a pattern match as indicated in patternOffset and pattern.
	patternOffset	If enableFramesize is set, this is the expected offset within the frame for the pattern match.
	patternOffset	If enableFramesize is set, this is the expected pattern for the pattern match.

TABLE 5-149. captureBuffer Sub-Commands

Member	Usage
get	Copies the data for a range of frame numbers from the hardware capture buffer. The high-level command, ixCheckTransmitDone , may be used to wait until all ports have finished transmitting. Note: For cards like 10GE LSMXM(4), LavaAP40/100GE2P, HSE40GE, and FlexAP40GE, this sub-command stops the capture process if it is still active.
getframe	Gets an individual frame's data.
clearConstraint	Clears the constraint values for jitter calculation.
setConstraint	Sets a new set of jitter calculation constraints.
getConstraint	Gets the current set of jitter calculation constraints.
getStatistics	Gets the jitter statistics for the current set of constraints.
export	Export the contents of a capture buffer for later import or usage by another program.
import	Import a previously saved and exported capture buffer for analysis.

The following example imports a previously saved capture buffer and print out the number of bytes in each frame:

```
captureBuffer import d:/adrian.cap 1 1 1
set numRxPackets [captureBuffer cget -numFrames]
ixPuts "$numRxPackets packets in buffer"
for {set frame 1} {$frame <= $numRxPackets} {incr frame} {
    captureBuffer getframe $frame
    set capframe [captureBuffer cget -frame]
    ixPuts "Frame $frame is [llength $capframe] bytes long"
}
```

Note. For some load modules (that is, LSM10GE), it is advisable to request `capturBuffer` data in chunks. Unless both the chassis and client machines have sufficiently high available memory, they may be overloaded by captured data.

qos

`qos` allows the user to set up the QoS counter filters and offsets. See [qos](#) on page A-604 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-150. qos Options

Member	Usage
<code>patternOffset</code>	The offset in the frame where a particular pattern is matched before QoS counting occurs.
<code>patternOffsetType</code>	
<code>patternMatch</code>	The value to look for at the <code>patternOffset</code> .
<code>patternMask</code>	The mask to be applied in the pattern match.
<code>byteOffset</code>	The offset in the packet where the priority value is located - to be used to increment the correct QoS counter.

TABLE 5-151. qos Sub-Commands

Member	Usage
<code>setup</code>	Sets the QoS counters for certain types of packets: <ul style="list-style-type: none"> • Ethernet II • 802.3 Snap • VLAN • PPP • Cisco HDLC

atmReassembly

The `atmReassembly` command is used to configure an ATM port to reassemble received data for particular VPI/VCIs. This is necessary if a receive port is to be used in an `atmStat` receive list or in `atmFilter`. Note that these commands automatically calls this command for the port, if it is not in the reassembly list. Except for receive ports using other than default encapsulation (`atmEncapsulationLLCRoutedCLIP`) in packet group mode, the `add` sub-command need never be called; the `del` and `removeAll` commands proves useful when changing a list. See [atmReassembly](#) on page A-37 for details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-152. atmReassembly Options

Member	Usage
<code>vpi</code>	The VPI and VCI to match.
<code>vci</code>	
<code>encapsulation</code>	The expected ATM encapsulation.
<code>enableIpTcpUdpChecksum</code>	If set, indicates that packets with this VPI/VCI pair are to be used in collecting TCP/UDP Checksum or QoS statistics.
<code>enableIpQos</code>	

TABLE 5-153. atmReassembly Sub-Commands

Member	Usage
add	Add or remove a particular VPI/VCI on a particular port to the reassembly list.
del	
removeAll	Remove all items from the reassembly list.
getFirstPair	Cycles through the VPI/VCI pairs in the list.
getNextPair	

atmFilter

The **atmFilter** command is used to set up capture/filter values for use with ATM ports. The frame data from one or more VPI/VCIs may be used to set the User Defined Statistics 1/2 (UDS 1, UDS 2), capture trigger or capture filter. The settings for a particular VPI/VCI on a port are set up with the command options and then memorized through the *set* sub-command. See [atmFilter](#) on page A-10 for details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-154. atmFilter Options

Member	Usage
enable	Enables or disables the use of a particular entry.
enableUds1	Selects one or more uses for the filter setup.
enableUds2	
enableFilter	
enableTrigger	
comparisonData	Establishes the data that is matched to satisfy the count,
comparisonMask	trigger, or filter function.

TABLE 5-155. atmFilter Sub-Commands

Member	Usage
set	Sets the options for a particular VPI/VCI on a particular port.
get	Gets the options for a particular VPI/VCI on a particular port.

stat

See the *Ixia Reference Guide* for a general discussion. Provides access to a wide range of statistics; the instantaneous value or rate is retrieved. See [stat](#) on page A-703 for full details. Statistics may be gathered in the following ways:

- Statistics in bulk, through the use of the `stat get allStats <chassis> <card> <port>` followed by calls to get the data using `stat cget -statName`.
- Rate statistics in bulk, through the use of the `stat getRate allStats <chassis> <card> <port>` followed by calls to get the data using `stat cget -statName`
- An individual statistic, through the use of the `stat get statName <chassis> <card> <port>`. The value is returned from the call.
- An individual rate statistic, through the use of the `stat getRate statName <chassis> <card> <port>`. The value is returned from the call.

Note also that most of the statistics are 64-bit values. `mpexpr` should be used to perform calculations on these values.

The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-156. stat Options

Member	Usage
mode	Sets the mode of the counters: <ul style="list-style-type: none"> • Normal. • QoS: Reuses eight of the counters for QoS values. • UDS 5,6: Reuses two of the counters for User Defined Statistics 5 and 6. • Checksum Errors: Reuses six hardware counters for IP, TCP, and UDP checksum errors. • Data Integrity: Reuses two counters for data integrity errors.
<statistics>	The number and type of statistics is too large to mention here. See stat on page A-703 for a description of the stat command and the <i>Ixia Reference Guide</i> for description of all statistics available.

TABLE 5-157. stat Sub-Commands

Member	Usage
get	Gets a particular statistic value or all statistics.
getRate	Gets the frame rate for a particular statistic value or all statistics.
getCaptureState	Determines whether a port's capture buffer is active or idle.
getLinkState	Gets the link state for a port.
getTransmissionState	Determines whether a port is actively transmitting or idle.
set	Sets the port's statistics mode as indicated in the mode member.

statGroup, statList and statWatch

These commands provide alternate means for accessing statistics across a set of ports. See [statGroup](#) on page A-741, [statList](#) on page A-743 and [statWatch](#) on page A-744 for full details. These commands are more efficient means of collecting multiple statistics or statistics from a group of ports.

A group of port may be formed using `statGroup` and all of the valid statistics for the ports in the group are available through `statList`.

As an alternative, `statWatch` may be used to set up a number of statistics watch sets. Each statistics watch has a unique ID and holds a list of ports and statistics. Once a stat watch is started, the indicated set of statistics is regularly retrieved for the indicated set of ports. `statList` is used to read the actual statistics.

Note also that most of the statistics are 64-bit values. `mpexpr` should be used to perform calculations on these values. The important options and sub-commands of `statGroup` are mentioned in the following table:

TABLE 5-158. statGroup Options

Member	Usage
<code>numPorts</code>	The current number of ports in the group.

TABLE 5-159. statGroup Sub-Commands

Member	Usage
<code>setDefault</code>	Resets the list to empty.
<code>add</code>	Adds a port to the group.
<code>del</code>	Deletes a specific port from the group.
<code>get</code>	Retrieves all of the valid statistics for all of the ports in the group. The individual statistics are available through <code>statList</code> .

The important options and sub-commands of `statList` are mentioned in the following table:

TABLE 5-160. statList Options

Member	Usage
<code><statistics></code>	The number and type of statistics is too large to mention here. See stat on page A-703 for a description of the stat command and the <i>Ixia Reference Guide</i> for description of all statistics available.

TABLE 5-161. statList Sub-Commands

Member	Usage
<code>get</code>	Gets a particular statistic value or all statistics.
<code>getRate</code>	Gets the frame rate for a particular statistic value or all statistics.

The important sub-commands of `statWatch` are mentioned in the following table:

TABLE 5-162. statWatch Sub-Commands

Member	Usage
<code>create</code>	Creates and destroys a stat watch.
<code>destroy</code>	
<code>addPort</code>	Adds or deletes a port to a particular stat watch.
<code>delPort</code>	
<code>addStat</code>	Adds or deletes a statistics to a particular stat watch.
<code>delStat</code>	

TABLE 5-162. statWatch Sub-Commands

Member	Usage
addStatRate	Adds or deletes a statistics rates to a particular stat watch.
delStatRate	
start	Starts and stops the stat watch process.
stop	

packetGroupStats

The *packetGroupStats* command is used to retrieve the statistics associated with packet groups, such as minimum latency, maximum latency and average latency. Some of the statistics are only available on specific types of ports; an attempt to read an unavailable statistic results in a error. Refer to the *Ixia Reference Guide* for list of which statistics are available.

Three sub-commands are used to retrieve the actual statistics.

- *packetGroupStats get chasID cardID portID [fromPGID toPGID]*: This fetches a range of statistics for the indicated port. The range is dictated by the *fromPGID* to the *toPGID*; if omitted, all PGIDs are retrieved, starting with PGID 0.
- *packetGroupStats getGroup index*: This fetches the statistics for a PGID that is *PGID = fromPGID + index*, where *fromPGID* is the value from the last call to *packetGroupStats get*. That is, *index = 0* refers to the *fromPGID* packet group ID.
- *packetGroupStats getFrameCount index*: Operates in the same manner as *getGroup*, with respect to the *index* parameter.

An additional feature available on some port types is the ability to collect latency measurements per packet group. The availability of this feature for a given port can be tested using the *port isValidFeature... portFeatureRxLatencyBin*. The port must be configured for wide packet groups (the *port's receiveMode* includes the *portRxModeWidePacketGroup* bit); the availability of this mode may be tested with *port isValidFeature... portFeatureRxWidePacketGroups*. (Note: When the port is in PRBS mode, all latency specific stats are removed.)

Latency bin dividing times must be set up with the *packetGroup*'s *enableLatencyBins, latencyBinList* option. Following a call to *packetGroupStats getGroup*, the *numLatencyBins* option is set and the latency bin information is available through calls to *getFirstLatencyBin*, *getNextLatencyBin* and *getLatencyBin*. The latency information is available in the options of the *latencyBin* command. Note that there is one more latency bin available than the number of dividers set in *packetGroup*'s *latencyBinList*, due to the implicit creation of a latency bin from the last divider to the maximum possible latency value.

An additional feature available on some port types is the ability to measure latency over time, per packet group. The availability of this feature for a given port can be tested using the *port isValidFeature... portFeatureRxTimeBin*. The port must be configured for wide packet groups (the *port's receiveMode* includes

the *portRxModeWidePacketGroup* bit); the availability of this mode may be tested with *port is ValidFeature... portFeatureRxWidePacketGroups*.

Time bins must be set up with the *packetGroup*'s *enableTimeBins*, *numPgidPerTimeBin*, *numTimeBins* and *timeBinDuration* options. Following a call to *packetGroupStats getGroup*, the *numTimeBins*, *numPgidPerTimeBin* and *timeBinDuration* options are set. Latency information for a particular time bin can be obtained by using the additional *timeBin* argument to the *getGroup* and *getGroupFrameCount* sub-commands.

See *packetGroupStats* on page A-482 for full details. The important options and sub-commands are mentioned in the following table:

TABLE 5-163. *packetGroupStats* options

Category	Member	Usage
Basic	numGroups	The number of actual groups received.
	totalFrames	The total number of frames used to calculate the statistics.
Latency	averageLatency minLatency maxLatency standardDeviation	The average/min/max latency for a group.
Latency Bins	numLatencyBins	The number of latency bins active.
Time Stamps	firstTimeStamp lastTimeStamp	First and last time stamp for packets in the packet group.

TABLE 5-163. packetGroupStats options

Category	Member	Usage
Rates	bitRate byteRate frameRate	The bit rate. The stats bitRate and byteRate are not available in Latency view when delay variation is specified as with <i>Latency Min Max Average</i> . Note: To get the valid frame rate, execute the packetGroupStats get command twice. In this case it is PG stats::packetGroupStats get \$chassisId \$cardId \$portId 0 \$ExpectedPgId . When you execute this command the first time, it returns 0. When you execute it the second time, it returns the valid count. This is because frame rate is calculated as per the difference in value between the current frame count and the previous frame count. The first time when you execute the packetGroupStats command, it will return the base value, which is the difference between current frame count and previous frame count. So you get the value as 0. The second time when you execute the command, it will calculate the frame rate by taking the difference between the base value and the current value.
PRBS	prbsBitsReceived prbsErroredBitgs prbsBerRatio	Per-PGID stats available when port is in PRBS mode

TABLE 5-164. packetGroupStats Sub-Commands

Member	Usage
get	Used to get the data for a range of group IDs into local memory.
getGroup	Used to retrieve the latency for a particular group.
getGroupFrameCount	Used to retrieve the number of frames for a group.
getFirstLatencyBin getNextLatencyBin getLatencyBin	Used to retrieve latency bin values to the <i>latencyBin</i> command's options.

latencyBin

This command holds the result of a *packetGroupStats getFirstLatencyBin/getNextLatencyBin/getLatencyBin* call. See *latencyBin* on page A-388 for full

details. The important options of this command are mentioned in the following table:

TABLE 5-165. latencyBin options

Category	Member	Usage
Basic	startTime	The start and stop times of the latency bin.
	stopTime	
	numFrames	The number of frames in the bin.
Latency	minLatency maxLatency	The min/max latency for a bin.
Time Stamps	firstTimeStamp lastTimeStamp	First and last time stamp for packets in the bin.
Rates	bitRate byteRate frameRate	The bit rate. Note that this requires multiple calls to <i>get</i> before valid values are obtained.

(Note: When the port is in PRBS mode, all latency specific stats are removed.)

vsrStat

`vsrStat` is used to retrieve statistics for VSR equipped 10GE cards. See [vsrStat](#) on page A-851 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-166. vsrStat options

Member	Usage
tx	Global transmit/receive statistics.
rx	
rxCodeWordViolationCounter	Receive statistics available on a per-channel basis.
rxLossOfSynchronization	
rxOutOfFrame	
rxCrcErrorCounter	

TABLE 5-167. vsrStat Sub-Commands

Member	Usage
get	Used to get all of the global and per channel statistics
getChannel	Used to fetch the channel specific statistics for one channel.

vsrError

`vsrError` is used to insert deliberate errors in VSR equipped 10Gigabit Ethernet cards. See [vsrStat](#) on page A-851 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-168. vsrError options

Member	Usage
enableChannelSwap	Controls features related to error detection and recovery.
enableDelimiterInsert	
enableProtectSwitch	
enableErrorCorrection	
bipErrorMask	Controls insertion of Section BIP errors.
bipErrorFrameCount	
bipInsertionMode	
crcErrorBlockCount	Controls insertion of CRC errors.
crcInsertionMode	
crcChannelSelection	
frameDelimiterErrorMask	Controls insertion of frame delimiter errors.
frameDelimiterErrorFrameCount	
frameDelimiterInsertionMode	
enableControlByte1	
enableControlByte2Ch1To6	
enableControlByte2Ch7To12	
enableControlByte3	
frameDelimiterControlByte1	
frameDelimiterControlByte2Ch1To6	
frameDelimiterControlByte2Ch7To12	
frameDelimiterControlByte3	
frameDelimiterChannelSelection	

TABLE 5-168. vsrError options

Member	Usage
channelSkewMode	Controls insertion of channel skew errors.
channelSkewDelayTime	
channelSkewInsertionMode	
channelSkewChannelSelection	
error8b10bCodeWordCount	Controls insertion of 8b/10b code word errors.
error8b10bCodeWordValue	
error8b10bInsertionMode	
error8b10bChannelSelection	
enableDisparityErrorCodeWord	
enableControlCharCodeWord	

TABLE 5-169. vsrError Sub-Commands

Member	Usage
insertError	Momentarily inserts a single instance of a particular error type.
start	Starts error insertion for all modes.
stop	Stops error insertion.

atmStat

The **atmStat** command is used to access statistics for particular VPI/VCI streams. VPI/VCIs for particular ports are added to a receive or transmit list with the *addRx* and *addTx* sub-commands. The statistics for all ports and VPI/VCIs in the lists are retrieved from the ports with the *get* sub-command. Individual statistics or rate statistics are accessed through the use of the *getStat* and *getRate* commands. The statistics are available in the command's options. See [atmStat](#) on page A-40 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-170. atmStat Options

Member	Usage
rxAtmCells	Statistics for receive ports.
rxAal5Frames	
rxAal5LengthErrors	
rxAal5TimeoutErrors	
txAtmCells	Statistics for transmit ports.
txAal5Bytes	
txAal5Frames	
txAal5ScheduledBytes	
txAal5ScheduledFrames	
vpi	The current VPI/VCI pair.
vci	

TABLE 5-171. atmStat Sub-Commands

Member	Usage
addRx	Adds a VPI/VCI for a particular port to the receive or transmit list.
addTx	
delRx	Deletes a VPI/VCI for a particular port from the receive or transmit list.
delTx	
removeAllRx	Clears all VPI/VCI pairs from the receive or transmit list for a particular port.
removeAllTx	
getFirstRxPair	Cycles through the receive or transmit lists.
getNextRxPair	
getFirstTxPair	
getNextTxPair	
get	Gets all of the statistics for all VPI/VCI pairs for all ports. Must be followed by a call to <i>getStat</i> or <i>getRate</i> .
getStat	Gets the statistics for a particular VPI/VCI on a particular port.
getRate	Gets the rate statistics for a particular VPI/VCI on a particular port.

streamTransmitStats

The **streamTransmitStats** command may be used to retrieve the per-stream transmit statistics. This may be checked through the use of the *portisValidFeature... portFeaturePerStreamTxStats* command. Per-stream transmit stats are retrieved by the stream id <number> per configuration on the port. They vary per port per transmit mode.

Note: The TXS8 supports 1 to 255 streams in packet stream transmit mode, and 1 to 128 streams in advanced mode.

StreamTransmitStats on ATM cards is limited to displaying statistics for 127 streams.

Statistics for a block of streams are retrieved through the use of the *get* command. Statistics for disabled streams are set to 0. Statistics for a particular stream are retrieved into the options of this command through the use of the *getGroup* command.

The *getGroup* command uses a ‘1’ based index into the block of streams fetched in the *get* command. For example, if *get* was used to fetch streams 101 through 200, then the statistics for stream 105 may be obtained by calling *getGroup* for index 5. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-172. streamTransmitStats Options

Member	Usage
numGroups	The number of groups retrieved by the <i>get</i> command.
frameRate	The command returns the rate at which the frames are sent. Note: To get the valid frame rate, execute the streamTransmitStats <i>get</i> command twice. In this case it is Stream stats::streamTransmitStats <i>get</i> \$chassisId \$cardId \$portId \$streamId. When you execute this command the first time, it returns 0. When you execute it the second time, it returns the valid count. This is because frame rate is calculated as per the difference in value between the current frame count and the previous frame count. The first time when you execute the streamTransmitStats command, it will return the base value, which is the difference between current frame count and previous frame count. So you get the value as 0. The second time when you execute the command, it will calculate the frame rate by taking the difference between the base value and the current value.
framesSent	The command returns the number of frames sent.
theoreticalAverageFrameRate	Calculates the long-term average frame rate for each stream

TABLE 5-173. **streamTransmitStats** Sub-Commands

Member	Usage
get	Fetches a block of data for a number of streams.
getGroup	Accesses a particular stream's statistics.

Interface Table

Several commands relate to the specification of interfaces and IP addresses.

- *protocolServer*: Enables various protocols.
- *Interface Table*: Constructs an table of interfaces, each interface contains a list of associated IPv4 and IPv6 addresses along with a MAC address.
- *IP*: Constructs an IP address to MAC address correspondence table.
- *Interface Table versus IP Address Table*: Discusses the differences and uses of the interface table versus the IP table.
- *sfpPlus*: Configures the small form-factor pluggable (SFP) transceiver interface, for NGY, and other 10GE load modules.

protocolServer

The **protocolServer** command is used to enable various protocols. See *protocolServer* on page A-582 for full details. The important options of this command are mentioned in the following table:

TABLE 5-174. protocolServer Options

Member	Usage
enableArpResponse	Enables ARP response.
enablePingResponse	Enables Ping response.
enableIgmpQueryResponse	Enables IGMP Query response.
enableOspfService	Enables OSPF service.
enableBgp4Service	Enables BGP4 service.
enableIsisService	Enables ISIS service.
enableRsvpService	Enables RSVP service.
enableRipService	Enables RIP service.
enableLdpService	Enables LDP service.
enableRipngService	Enables RIPng service.
enableMldService	Enables MLD service.
enableOspfv3Service	Enables OSPFv3 service.
enablePimsmService	Enables PIM SM service.
enableBgp4CreateInterface	Enables BGP4 create interface.
enableIsisCreateInterface	Enables ISIS create interface.
enableOspfCreateInterface	Enables OSPF create interface.
enableRipCreateInterface	Enables RIP create interface.

TABLE 5-174. protocolServer Options

Member	Usage
enableRsvpCreateInterface	Enables RSVP create interface.
enableIgmpCreateInterface	Enables IGMP create interface.

Interface Table

The interface table is used to hold a number of logical interfaces that are associated with an Ixia port. Each interface may have none or more IPv4¹ and IPv6 addresses associated with a MAC address and optional VLAN ID.

Refer to the *IxNetwork Users Guide* for a discussion of the Ixia Protocol Server's testing model with respect to interfaces.

Following are the interface table related commands:

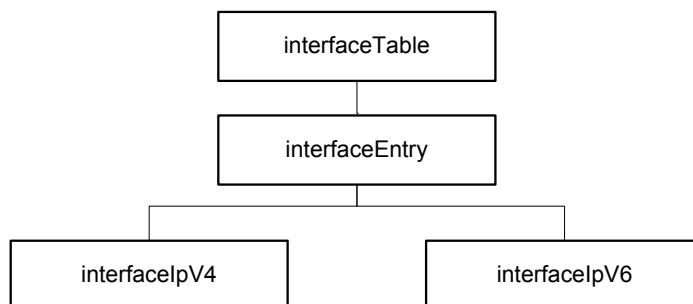
- *interfaceTable*: Holds a list of interfaces for a port.
- *interfaceEntry*: Holds a list of IPv4/IPv6 addresses associated with an interface.
- *interfaceIpV4*: An individual IPv4 address and related parameters.
- *interfaceIpV6*: An individual IPv6 address and related parameters.

In addition, the IP addresses associated with an interface as well as perceived neighbors may be accessed with the following commands:

- *discoveredList*
- *discoveredAddress*
- *discoveredNeighbor*

These commands, and the data that they maintain are arranged in a hierarchy, as shown in *Figure 5-2, “Interface Table Command Hierarchy,” on page 5-109*.

Figure 5-2. Interface Table Command Hierarchy



1. Only one IPv4 is currently allowed for interfaces.

interfaceTable

The *interfaceTable* command is used to configure interfaces associated with a port. Interfaces hold *interfaceEntry* elements, each of which includes multiple IPv4 and IPv6 addresses. Note that the *select* command must be used before any other sub-commands to indicate the chassis, card, and port in use. Refer to *interfaceTable* on page A-305 for full details. The typical series of operations are mentioned in the following table:

TABLE 5-175. Typical Interface Table Operations

Operation	Steps
add interface table items	<ol style="list-style-type: none"> 1. Use the <i>select</i> sub-command of the <i>interfaceTable</i> command to select the port being modified. 2. Set IPv4 or IPv6 address values using the <i>interfaceIpV4</i> or <i>interfaceIpV6</i> command. DHCP may be configured in the <i>dhcpV4Properties</i> command. DHCPv6 may be configured in the <i>dhcpV6Properties</i> command. 3. Use the <i>addItem</i> sub-command of the <i>interfaceEntry</i> command; which adds the address to an individual interface entry. 4. Repeat steps 2 and 3 to add IPv4 and/or IPv6 addresses to the interface. 5. Set the <i>enable</i>, <i>description</i>, <i>macAddress</i> and <i>VLAN</i> attributes for the interface entry. 6. Use the <i>addInterface</i> sub-command of the <i>interfaceTable</i> command to add the interface entry to the table. 7. Repeat steps 2 through 6 to add additional interfaces to the interface table for a port.
look through interface table	<ol style="list-style-type: none"> 1. Use the <i>select</i> sub-command of the <i>interfaceTable</i> command to select the port being modified. 2. Use the <i>getFirstInterface</i> sub-command of the <i>interfaceTable</i> command to reference the first interface entry in the list. 3. Use the <i>getFirstItem</i> sub-command of the <i>interfaceEntry</i> command to get to the first interface entry. Note that separate lists are maintained for IPv4 and IPv6 addresses. 4. Use the options in the <i>interfaceIpV4</i> or <i>interfaceIpV6</i> command to look at the IPv4 or IPv6 address data. 5. Use the <i>getNextItem</i> sub-command of the <i>interfaceEntry</i> command to position to the next address in the interface entry. 6. Use the <i>getNextInterface</i> sub-command of the <i>interfaceTable</i> command to reference the next interface entry in the list.

TABLE 5-175. Typical Interface Table Operations

Operation	Steps
	<p>7. Repeat steps 3 and 4 to look through the IPv4/IPv6 addresses in the interface entry.</p>
	<p>8. Repeat steps 6 to loop through all defined interfaces until an error is returned.</p>

The important sub-commands of this command are mentioned in the following table:

TABLE 5-176. interfaceTable Sub-Commands

Member	Usage
select	Sets the chassis, card and port that are operated on by the remaining sub-commands. This sub-command must be used first.
clearAllInterfaces	Clears the interface table.
addInterface	Adds the interface entry described in the <i>interfaceEntry</i> command to the interface table at the current table position.
delInterface	Deletes the interface table item at the current position or matched by a description.
getInterface	Finds the interface table item for a particular interface description.
getFcoeDiscoveredInfo	Gets the FCoE assigned address and other information which matches the specified description.
getFirstInterface	Positions to the first interface table item.
getNextInterface	Moves to the next interface table item.
sendRouterSolicitation	Sends a router solicitation request to the link. Routers on the link sends back router announcement messages that is accessible in the discovered table.
clearDiscoveredNeighborTable	Clears all of the discovered neighbors.
requestDiscoveredTable getDiscoveredList getDhcpV4DiscoveredInfo getDhcpV6DiscoveredInfo getFcoeDiscoveredInfo getPtpDiscoveredInfo	These commands are used in sequence to retrieve the discovered neighbor and address tables. This data is accessed through the <i>discoveredList</i> , <i>dhcpV4DiscoveredInfo</i> , <i>dhcpV6DiscoveredInfo</i> , <i>fcoeDiscoveredInfo</i> and <i>ptpDiscoveredInfo</i> commands.
sendArp	Sends an ARP request to one or all enabled interfaces. Results are read back through <i>getDiscoveredList</i> .
sendArpClear	Clears the ARP table for one or all enabled interfaces.

TABLE 5-176. interfaceTable Sub-Commands

Member	Usage
sendArpRefresh	Re-reads the ARP data from the port CPU.
setInterface	Sets an interface entry in the interface to the specified description.

interfaceEntry

Interface entries hold one or more IPv4 or IPv6 addresses; the *interfaceTable* takes care of keeping the actual list of interfaces. DHCP and DHCPv6 for IPv4 may also be enabled. See [interfaceEntry](#) on page A-299 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-177. interfaceEntry Options

Member	Usage
enable	Enables the use of the interface entry.
enableDhcp	Enables the use of DHCP for IPv4 addresses on the port. DHCP parameters are configured using the dhcpV4Properties command.
enableDhcpV6	Enables the use of DHCPv6 for IPv4 addresses on the port. DHCP parameters are configured using the dhcpV6DiscoveredInfo command.
enableFlogi	Enable Fabric login (for FCoE protocol)
description	A description of the interface. This description is used for matching in the <i>interfaceTable</i> 's <i>delItem</i> and <i>getItem</i> commands.
interfaceType	The type of the interface: connected, routed, GRE, NPIV, or PTP.
ipv6 gateway	There can be one gateway per IPv6 interface.
macAddress	The MAC address associated with the interface entry.
enableVlan	Enable VLAN encapsulation for the interface.
vlanId	If <i>enableVlan</i> is set, this is the ID used for the VLAN encapsulation.
eui64Id	The EUI-64 ID associated with POS boards with IPv6 support.
greSourceIpAddress greDesIpAddress enableGreChecksum enableGreSequence enableGreKey greInKey greOutKey	The settings for GRE headers, when <i>interfaceType</i> is GRE.

TABLE 5-178. interfaceEntry Sub-Commands

Member	Usage
clearAllItems	Clears all IPv4/IPv6 addresses. Two separate lists are used for each of the IPv4 and IPv6 addresses.
addItem	Adds an IPv4/IPv6 to the respective list in the entry. Two separate lists are used for each of the IPv4 and IPv6 addresses.
delItem	Deletes the currently addressed item, or one that matches a particular address.
getFirstItem	References the first item in one of the lists.
getNextItem	References the next item in one of the lists.
getItem	Refers to a particular item matched by a particular address.

interfaceIpV4

The *interfaceIpV4* holds a single IPv4 address and related data. It is added to one of the lists in the *interfaceEntry* using the *interfaceEntry addItem* command. Refer to [interfaceIpV4](#) on page A-303 for full details. The important options of this command are mentioned in the following table:

TABLE 5-179. interfaceIpV4 Options

Member	Usage
ipAddress	The IPv4 address.
gatewayIpAddress	The gateway address for the address.
maskWidth	The size of the mask for the address, counting from the high-order bit.

interfaceIpV6

The *interfaceIpV6* holds a single IPv6 address and data. It is added to one of the lists in the *interfaceEntry* using *interfaceEntry addItem*. Refer to [interfaceIpV6](#) on page A-304 for full details. The important options of this command are mentioned in the following table:

TABLE 5-180. interfaceIpV6 Options

Member	Usage
ipAddress	The IPv6 address.
maskWidth	The size of the mask for the address, counting from the high-order bit.

discoveredList

The *discoveredList* command must be preceded with use of three commands in the *Interface Table* command: *sendRouterSolicitation*, *requestDiscoveredTable*,

and *getDiscoveredList*. The *discoveredList* command is used to look through two lists associated with an interface, as follows:

- Neighbor list: Contains a list of discovered neighbors, each of which contains a MAC address and a list of IP addresses.
- Address list: Contains the list of IP addresses associated with the interface.

Refer to *discoveredList* on page A-144 for full details. The important sub-commands of this command are mentioned in the following table:

TABLE 5-181. discoveredList Sub-Commands

Member	Usage
getFirstAddress getNextAddress	Loops through the IP addresses assigned to the interface. The IP address itself is accessed through the use of the <i>discoveredAddress</i> command.
getFirstNeighbor getNextNeighbor	Loops through the neighbors found for the interface. The neighbor's information itself is accessed through the use of the <i>discoveredNeighbor</i> command.

discoveredAddress

The *discoveredAddress* command holds an IPv4 or IPv6 address associated with an interface (as retrieved in *discoveredList*) or the IPv4/IPv6 address associated with a neighbor (as retrieved in *discoveredNeighbor*). Refer to *discoveredAddress* on page A-143 for full details. The important options of this command are mentioned in the following table:

TABLE 5-182. discoveredAddress Options

Member	Usage
ipAddress	The IPv4 or IPv6 address.

discoveredNeighbor

The *discoveredNeighbor* command holds an entry for each neighbor discovered as a result of router discovery or neighbor discovery announcements. Each neighbor entry has the following:

- MAC address: The MAC address of the discovered interface.
- Router flag: If the neighbor is a router.
- Address list: A list of IP addresses associated with the neighbor's interface, accessed with the *discoveredAddress* command.

The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-183. discoveredNeighbor Options

Member	Usage
macAddress	The MAC address associated with the neighbor.
isRouter	If <i>true</i> , indicates that the neighbor is a router.

TABLE 5-184. discoveredNeighbor Sub-Commands

Member	Usage
getFirstAddress	Loops through the IP addresses associated with the neighbor.
getNextAddress	The IP address itself is accessed through the use of the <i>discoveredAddress</i> command.

Using DHCP with Interfaces

A DHCP client may be enabled on intelligent ports and then used for the source addresses in stream traffic. The steps necessary to accomplish these are as follows:

- *interfaceEntry*: Set the *enableDhcp* flag to enable the use of DHCP.
- *dhcpV4Properties*: Set the DHCP negotiation properties.
 - *dhcpV4Tlv*: Can be used to set up DHCP properties beyond those exposed in *dhcpV4Properties*.
- *interfaceTable*: Use *addInterface* to add the DHCP enabled interface to the port.
- *ixWritePortsToHardware*: Or similar command to send the configuration to the port(s).
- *interfaceTable*: Use the *requestDiscoveredTable* followed by the *getDhcpV4DiscoveredInfo* sub-commands to read back the assigned DHCP address. This information is available in the following commands:
 - *dhcpV4DiscoveredInfo*: Allows access to the assigned address and other common parameters.
 - *dhcpV4Tlv*: Allows access to all other parameters as TLVs.
- *stream*: Set the *enableSourceInterface* and *sourceInterfaceDescription* fields to specify that the MAC and IPv4 addresses should be taken from a particular interface entry.

dhcpV4Properties

The *dhcpV4Properties* command allows you to set the most frequently used DHCP parameters to be used in negotiation with a DHCP server. The values in this command are applied to an *interfaceEntry* being added to the *interfaceTable*. Other DHCP parameters may be set with the *dhcpV4Tlv* command. Refer to

dhcpV4Properties on page A-135 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-185. dhcpV4Properties Options

Member	Usage
clientId	The client identifier. If "", then the MAC address of the interface is used.
serverId	If specified, only the indicated DHCP server is used. Otherwise, any available DHCP server is used.
vendorId	The vendor ID for the client.
renewTimer	The client's desired renewal time. The lesser of this time and the DHCP server's response is used.

TABLE 5-186. dhcpV4Properties Sub-Commands

Member	Usage
addTlv	Associates the values in <i>dhcpV4Tlv</i> with this DHCP property set.
getFirstTlv	Fetches a TLV value either by iterating through all the items, or by index.
getNextTlv	
getTlv	
delTlv	Removes a single TLV or all TLVs.
removeAllTlvs	

dhcpV4DiscoveredInfo

The *dhcpV4DiscoveredInfo* command makes the frequently used DHCP parameters negotiated with a DHCP server available to you. Other DHCP parameters may be read with the *dhcpV4Tlv* command. Refer to *dhcpV4DiscoveredInfo* on page A-133 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-187. dhcpV4DiscoveredInfo Options

Member	Usage
ipAddress	The DHCP server assigned IPv4 address.
gatewayIpAddress	The DHCP server assigned gateway address.
prefixLength	The prefix/mask length for the network associated with the <i>ipAddress</i> .
leaseDuration	The server's desired renewal time. The lesser of this time and the client's desired value is used.

TABLE 5-188. dhcpV4DiscoveredInfo Sub-Commands

Member	Usage
getFirstTlv	Fetches a TLV value either by iterating through all the items, or by index.
getNextTlv	
getTlv	

dhcpV4Tlv

The *dhcpV4Tlv* command is used to set DHCP options used by the client in the DHCP negotiation and to read the results from the DHCP server. The client values are set with the *dhcpV4Properties addTlv* sub-command. The server values are read with the *dhcpV4DiscoveredInfo get*Tlv* sub-commands. Refer to *dhcpV4Tlv* on page A-137 for full details. The important options of this command are mentioned in the following table:

TABLE 5-189. **dhcpV4Tlv Options**

Member	Usage
type	The type number of the DHCP parameter.
value	The value of the DHCP parameter. The length is inferred from the length of this string.

Using DHCPv6 with Interfaces

A DHCPv6 client may be enabled on intelligent ports and then used for the source addresses in stream traffic. The following steps are necessary to accomplish this:

- *interfaceEntry*: Set the *enableDhcpV6* flag to enable the use of DHCP.
- *dhcpV6DiscoveredInfo*: Set the DHCPv6 negotiation properties.
- *dhcpV6Tlv*: Can be used to set up DHCPv6 properties beyond those exposed in *dhcpV6Properties*.
- *interfaceTable*: Use *addInterface* to add the DHCPv6 enabled interface to the port.
- *ixWritePortsToHardware*: Or similar command to send the configuration to the port(s).
- *interfaceTable*: Use the *requestDiscoveredTable* followed by the *getDhcpV6DiscoveredInfo* sub-commands to read back the assigned DHCPv6 address. This information is available in the following commands:
 - *dhcpV6DiscoveredInfo*: Allows access to the assigned address and other common parameters.
 - *dhcpV6Tlv*: Allows access to all other parameters as TLVs.
- *stream*: Set the *enableSourceInterface* and *sourceInterfaceDescription* fields to specify that the MAC and IPv4 addresses should be taken from a particular interface entry.

dhcpV6Properties

The *dhcpV6Properties* command allows you to set the most frequently used DHCPv6 parameters to be used in negotiation with a DHCP server. The values in this command are applied to an *interfaceEntry* being added to the *interfaceTable*. Other DHCP parameters may be set with the *dhcpV6Tlv* command. Refer to

dhcpV6Properties on page A-140 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-190. dhcpV6Properties Options

Member	Usage
iaID	The client identifier, which must be unique for the subnet that the interface is connected to
iaType	The type of DHCPv6 address.
renewTimer	The requested value for the renewal time, in seconds.

TABLE 5-191. dhcpV6Properties Sub-Commands

Member	Usage
addTlv	Associates the values in <i>dhcpV6Tlv</i> with this DHCP property set.
getFirstTlv	Fetches a TLV value either by iterating through all the items, or by index.
getNextTlv	
getTlv	
delTlv	Removes a single TLV or all TLVs.
removeAllTlvs	

dhcpV6DiscoveredInfo

The *dhcpV6DiscoveredInfo* command makes the frequently used DHCPv6 parameters negotiated with a DHCP server available to you. Other DHCPv6 parameters may be read with the *dhcpV6Tlv* command. Refer to *dhcpV6DiscoveredInfo* on page A-138 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-192. dhcpV6DiscoveredInfo Options

Member	Usage
discoveredAddressList	A list of discovered IP addresses.
iaRebindTime	The rebind timer value specified by the DHCPv6 Server, in seconds
iaRenewTime	The renew timer value specified by the DHCPv6 Server, in seconds

TABLE 5-193. dhcpV6DiscoveredInfo Sub-Commands

Member	Usage
getFirstTlv	Fetches a TLV value either by iterating through all the items, or by index.
getNextTlv	
getTlv	

dhcpV6Tlv

The *dhcpV6Tlv* command is used to set DHCPv6 options used by the client in the DHCPv6 negotiation and to read the results from the DHCP server. The client values are set with the *dhcpV6Properties addTlv* sub-command. The server values are read with the *dhcpV6DiscoveredInfo get*Tlv* sub-commands. Refer to *dhcpV6Tlv* on page A-142 for full details. The important options of this command are mentioned in the following table:

TABLE 5-194. *dhcpV4Tlv* Options

Member	Usage
type	The type number of the DHCP parameter.
value	The value of the DHCP parameter. The length is inferred from the length of this string.

Using PTP with Interfaces

Precision Time Protocol (PTP) enables precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing, and distributed objects. See *ptp* on page A-586 for details. Associated commands include *ptpAnnounce* on page A-592, *ptpDelayRequest* on page A-595, *ptpDelayResponse* on page A-596, *ptpDiscoveredInfo* on page A-597, *ptpFollowUp* on page A-599, *ptpProperties* on page A-600, and *ptpSync* on page A-603.

Using Fibre Channel and FCoE**fibreChannel**

The *fibreChannel* command supports FCoE header and trailer in streams. See *fibreChannel* on page A-199 for details.

fcoe

The *fcoe* command is used to configure Fibre Channel over Ethernet (FCoE) header and trailer packet. FCoE is a method of communicating data for streams and protocols. See *fcoe* on page A-162 for details. Associated commands include *fcoeDiscoveredInfo* on page A-174 and *fcoeProperties* on page A-179.

To configure an unconnected NPIV interface, see also *npiProperties* on page A-427.

IP**ipAddressTable**

The address table is a list of entries, each of which is described in the item command. The address table command is used to position within the list and

elements are accessed with the list object. The typical series of operations are mentioned in the following table:

TABLE 5-195. Typical Address Table Operations

Operation	Steps
add address table items	<ol style="list-style-type: none"> 1. Set values in the <code>ipAddressTableItem</code> command. 2. Use the <code>set</code> sub-command of the <code>ipAddressTableItem</code> command which transfers the data into a holding area. 3. Use the <code>addItem</code> sub-command of the <code>ipAddressTable</code> command to move the data from the holding area to the actual list. 4. Repeat steps 1, 2 and 3 for each table item to be added. 5. Use the <code>set</code> sub-command of the <code>ipAddressTable</code> command to send the table to the hardware.
look through address table	<ol style="list-style-type: none"> 1. Use the <code>get</code> sub-command <code>ipAddressTable</code> command to transfer the data from the hardware to the object. 2. Use the <code>get</code> sub-command of the <code>ipAddressTableItem</code> command to get the data into the <code>ipAddressTableItem</code> options. 3. Use the <code>getNextItem</code> sub-command of the <code>ipAddressTable</code> command to position to the next table item. 4. Repeat steps 2 and 3 until an error is returned from step 3.
find the address table item for an IP address	<ol style="list-style-type: none"> 1. Use the <code>getItem</code> sub-command of the <code>ipAddressTable</code> command to position the list to the correct entry. 2. Use the <code>get</code> sub-command of the <code>ipAddressTableItem</code> command to get the data into the <code>ipAddressTableItem</code> options.

See [ipAddressTable](#) on page A-331 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-196. ipAddressTable Options

Member	Usage
defaultGateway	The IP address of where all non-overridden ARP requests are sent. This is usually the address of the Device Under Test.

TABLE 5-197. ipAddressTable Sub-Commands

Member	Usage
clear	Clears the IP address table.
addItem	Adds the address table item as set by the last call to <code>ipAddressTableItem.set</code> to the address table at the current table position.
delItem	Deletes the address table item at the current position.
getItem	Finds the address table item for a particular IP address.

TABLE 5-197. ipAddressTable Sub-Commands

Member	Usage
getFirstItem	Positions to the first address table item.
getNextItem	Moves to the next address table item.

ipAddressTableItem

This command holds an individual table item; `ipAddressTable` takes care of keeping the actual list of address table items. See [ipAddressTableItem](#) on page A-333 for full details. The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-198. addressTableItem Options

Member	Usage
fromIpAddress toIpAddress	The IP address range.
fromMacAddress toMacAddress	The MAC address range.
numAddresses	The number of consecutive addresses.
enableUseNetwork netMask	Enables and sets the network mask to be applied to the IP addresses.
overrideDefaultGateway	Override the default gateway address from the <code>protocolServer</code> object.
gatewayIpAddress	If the gateway is overridden, this is the new gateway address value.
mappingOption	IP to MAC mapping: Either one IP to one MAC or many IP to one MAC.
enableVlan vlanId	Enable VLAN encapsulation for the routing protocols, using a specified VLAN ID. See Interface Table on page 5-108 for a list of protocols that may be VLAN encapsulated.

TABLE 5-199. addressTableItem Sub-Commands

Member	Usage
get	Gets the current item from the <code>ipAddressTable</code> command.
set	Saves the current item for use by <code>ipAddressTable</code> .

**Interface Table
versus IP Address
Table**

The interface table is a new means of associating IPv4 and IPv6 addresses with ports and eventually replaces the IP table. Observe the following rules:

- Write new tests using the interface table.

- Do not mix interface table and IP table usage for the same test. If interface tables are available, the software uses them exclusively. If no interface tables are present, the data in the IP table is used automatically.
- Interface tables must be used for some of the new, advanced protocol tests. These include the following:
 - RIPng
 - OSPF
 - ISIS
 - BGP when used with any TXS or 10GE
- Continue to use IP tables when a large number of IP addresses are used, for example in ARP testing.

sfpPlus

The **sfpPlus** command is used to configure the small form-factor pluggable (SFP) transceiver interface that was added to NGY and other 10GE load modules. See [sfpPlus](#) on page A-655 for full details.

The important options and sub-commands of this command are mentioned in the following table:

TABLE 5-200. sfpPlus Options

Member	Usage
enableMonitorLos	Enable monitor SFP Loss of Signal.
enableMonitorModuleReadySignal	Enable monitor SFP Module Ready Signal.
enableAutomaticDetect	Enable automatic detection of transceiver type.
type	Configure the transceiver type.

Port CPU Control

Three commands are available which relate to port CPU operation and are covered in the next three sub-sections.

Port CPU Control

Each Ixia port that has a local CPU may be reset through the use of the *portCPU reset* command. Refer to *portCpu* on page A-555 for a full description of this command.

The *port* command's *isValidFeature* sub-command may be used to determine if a given port has a CPU. Use the following sequence:

```
if [port isValidFeature $chassis $card $port portFeatureLocalCPU] {  
    ... port has a CPU ...  
}
```

The important sub-commands and options of this command are mentioned in the following table:

TABLE 5-201. **portCpu Sub-Commands**

Member	Usage
reset	Causes the port to reboot its operating system and return to its initial state. Any optional loaded packages are removed. The current port and stream configuration is not affected.

TABLE 5-202. **portCpu Options**

Member	Usage
memory	The amount of memory, expressed in MB, associated with the port.

Issue Port CPU Command

Most intelligent Ixia port runs the Linux Operating system. Any Linux command may be remotely executed by TCL programming. The *port* command's *isValidFeature* sub-command may be used to determine if a given port runs Linux. Use the following sequence:

```
if [port isValidFeature $chassis $card $port portFeatureIxRouter] {  
    ... port runs Linux ...  
}
```

pcpuCommandService

The *pcpuCommandService* command allows commands to be sent to a set of ports and executed simultaneously. Different commands may be executed on different ports.

Refer to [pcpuCommandService](#) on page A-504 for a full description of this command. The important sub-commands and options of this command are mentioned in the following table:

TABLE 5-203. pcpuCommandService Sub-Commands

Member	Usage
add	Adds a command to a specific port's list.
del	Deletes a command from a specific port's list.
execute	Sends commands to all ports and executes them.
getFirst/getNext	Cycles through list of ports and commands. Command results may be retrieved.

TABLE 5-204. pcpuCommandService Options

Member	Usage
chassisID/cardID/portID	The port being addressed.
command	The command to be executed (add) or executed (get).
input	Optional lines of text that is used as the command's standard input stream.
output	The text from standard output from the executed command. A maximum of 1024 bytes is saved.
error	The text from standard error from the executed command. A maximum of 1024 bytes is saved.
result	The result code of a command's execution. In general zero means no error and a non-zero indicates an error.

A high-level command, [issuePcpuCommand](#), is also offered.

serviceManager

Most intelligent Ixia ports run the Linux Operating system. Software may be developed for these ports using the guidelines documented in the *Ixia Linux SDK Guide*. Such software must be combined in a set of files called a *package* and downloaded to a set of ports. Software packages must have been previously placed on the chassis associated with any affected port, in the following folder:

C:\Program Files\Ixia\packages

The following files constitute a package:

- A control file: This file, with the extension *ini*, allows different data files to be downloaded to the ports based on the type of port processor and operating system version. All *ini* files must be located in the *C:\Program Files\Ixia\packages* folder.
- Data files: One or more data files, each specifically compiled for a specific CPU type and/or operating system. Data files are typically organized in separate folders:

```
<package>/<processor>/package.tgz
```

For example, a package named *sample* which supports the PPC 405 and 750 processors would have the following files:

```
C:\Program Files\Ixia\packages\sample.ini
C:\Program Files\Ixia\packages\sample\ppc405\sample.tgz
C:\Program Files\Ixia\packages\sample\ppc750\sample.tgz
```

Control File Format

Each package must have a control file named *<package>.ini*, where *<package>* is a unique name. The following types of statements are allowed in a control file:

- **autoload.** This is a single, optional statement:

```
autoload=1
```

If present, this indicates that the associated package is to be loaded onto all port CPUs and started as per the indicated by the packages *start.sh* file, discussed below. This statement should only be used if it really necessary for a package to permanently reside on a port.

- **package.** A package statement is of the form:

```
package [name=value, [...] path=<package path>
```

One or more *name=value* pairs may be used to qualify the condition under which a particular version of the software is used. The possible *name* values are:

- **processor.** This is matched against the type of CPU running on the port. The legal values are as follows:

- ppc405: Power PC, model 405.
- ppc750: Power PC, model 750.
- sh4: Hitachi SH4.

- **platform.** This is matched against the version of IxOS software running on the port. The legal values take the following forms:

- *version*: This version only.
- *-version*: Up to and including this version.
- *version-*: From this version on.

A *version* is of the form *n.n[n...J*. That is, two or more decimal separated numbers. For example, 3.65 or 3.70.24.9.

An example *ini* file is shown below:

```
#sample.ini
package processor=ppc405, path=sample/ppc405/sample.tgz
package processor=ppc750, platform=3.65-3.70, path=sample/
    ppc750/sampleOld.tgz
package processor=ppc750, platform=3.80, path=sampleNew.tgz
```

Data Files

The data files associated with a package are contained in a single gzipped tar file. The Linux command line to create such a package is as follows:

```
tar -czf <package>.tgz <file1> <file2> ...
```

The files in the tar file is unpacked on each processor to the following:

```
/opt/<package>
```

folder. The following types of files are contained in the tar file:

- **start.sh**: This file is mandatory and describes how to install and start the package. It is automatically run as soon as the package is downloaded to a port by `/bin/sh start.sh` from `/opt/<package>`. For example, if a tar file contained the following files:

- start.sh
- stop.sh
- bin/sample
- lib/libsample.so

Then an appropriate `start.sh` would be:

```
#start.sh
# Symlink to /bin and /lib
ln -s ../../opt/sample/bin/sample ../../bin
ln -s ../../opt/sample/lib/libsample.so ../../lib

/bin/sample > /dev/console 2>&1
```

- **stop.sh**: This optional file is run by the service manager prior to deleting the package's files in `/opt/<package>`. This script should kill any processes that the `start.sh` script spawned and remove any files that were installed outside of `/opt/<package>`.
- **executable files**: Although the files may be organized in any manner, we suggest that the package's main executable be placed in a `bin` folder and that any library be placed in a `lib` folder.

serviceManager

The `serviceManager` command is used to download and manage packages. Refer to [serviceManager](#) on page A-650 for a complete explanation of these sub-commands. Note the `serviceManager` command is valid in Windows based environments.

The important sub-commands of this command are mentioned in the following table:

TABLE 5-205. serviceManager Sub-Commands

Member	Usage
downloadPackage	Downloads and starts a package to the ports associated with a port group. The port group is built using the <i>portGroup</i> commands.
deletePackage	Stops and deletes a package from the ports associated with a port group.
getInstalledPackages	Returns a list of packages installed on a particular port.



A

IxTclHAL Commands

NAME - arp

arp - configure the ARP parameters on a stream of a port.

SYNOPSIS

arp *sub-command options*

DESCRIPTION

The **arp** command is used to configure the ARP parameters on a stream of a port to transmit ARP frames. Any number of varying ARP frames may be generated.

STANDARD OPTIONS

destHardwareAddr

The MAC address of the interface receiving the ARP message. (*default = 00 de bb 00 00 01*)

destHardware AddrMode

Indicates how the *destHardwareAddr* field is to vary between consecutive frames.

Option	Value	Usage
<i>arpIdle</i>	0	(<i>default</i>) no change
<i>arpIncrement</i>	1	increment by 1 for the count in <i>destHardwareAddrRepeatCount</i> .
<i>arpDecrement</i>	2	decrement by 1 for the count in <i>destHardwareAddrRepeatCount</i> .
<i>arpContinuousIncrement</i>	3	increment by 1 continuously.
<i>arpContinuousDecrement</i>	4	decrement by 1 continuously.

destHardwareAddr RepeatCount

Indicates the repeat count for the *destHardwareAddrMode* increment and decrement options. (*default = 0*)

destProtocolAddr

Protocol address of the station that is receiving the ARP message. (*default = 127.0.0.1*)

destProtocolAddrMode

Indicates how the *destProtocolAddr* field is to vary between consecutive frames.

Option	Value	Usage
<i>arpIdle</i>	0	(<i>default</i>) no change
<i>arpIncrement</i>	1	increment by 1 for the count in <i>destProtocolAddrRepeatCount</i> .
<i>arpDecrement</i>	2	decrement by 1 for the count in <i>destProtocolAddrRepeatCount</i> .
<i>arpContinuousIncrement</i>	3	increment by 1 continuously.
<i>arpContinuousDecrement</i>	4	decrement by 1 continuously.

destProtocolAddr RepeatCount

Indicates the repeat count for the *destProtocolAddrMode* increment and decrement options. (*default = 0*)

hardwareAddrLength

Read-Only. Number of bytes in the hardware address. (*default = 6*)

hardwareType

Read-Only. Indicates the hardware type that the physical layer of the network is using. Available option values are mentioned in the following table:

Option	Value	Usage
<i>hwTypeEthernet</i>	1	(default) Ethernet 10 Mb
<i>hwTypeAmateur</i>	3	Amateur radio AX.25
<i>hwTypeProteon</i>	4	Proteon ProNET token ring
<i>hwTypeChaos</i>	5	Chaos
<i>hwTypeIEEE</i>	6	IEEE 802 networks
<i>hwTypeARCNET</i>	7	ARCNET
<i>hwTypeHyperchannel</i>	8	Hyperchannel
<i>hwTypeLocalTalk</i>	11	LocalTalk

operation

The type of operation the ARP process is attempting. Available options are mentioned in the following table:

option	Value	Usage
<i>arpRequest</i>	1	(default) ARP request
<i>arpReply</i>	2	ARP reply or response
<i>rarpRequest</i>	3	RARP request
<i>rarpReply</i>	4	RARP reply or response

protocolAddrLength

Read-Only. Number of bytes that each of the protocol addresses, source and target, contains in the ARP frame. (default = 4)

protocolType

Read-Only. Indicates the type of network protocol address the local network (or subnet) is using. (default = 0x0800)

sourceHardwareAddr

The MAC address of the sending ARP interface. (default = 00 de bb 00 00 00)

**sourceHardware
AddrMode**

Indicates how the *sourceHardwareAddr* field is to vary between consecutive frames.

Option	Value	Usage
<i>arpIdle</i>	0	(default) no change
<i>arpIncrement</i>	1	increment by 1 for the count in <i>sourceHardwareAddrRepeatCount</i> .
<i>arpDecrement</i>	2	decrement by 1 for the count in <i>sourceHardwareAddrRepeatCount</i> .
<i>arpContinuousIncrement</i>	3	increment by 1 continuously.
<i>arpContinuousDecrement</i>	3	decrement by 1 continuously.

**sourceHardware
AddrRepeatCount**

Indicates the repeat count for the *sourceHardwareAddrMode* increment and decrement options. (default = 0)

sourceProtocolAddr

Protocol address of the station that is sending the ARP message. (default = 127.0.0.1)

**sourceProtocol
AddrMode**

Indicates how the *sourceProtocolAddr* field is to vary between consecutive frames.

Option	Value	Usage
<i>arpIdle</i>	0	(default) no change

Option	Value	Usage
<i>arpIncrement</i>	1	increment by 1 for the count in <i>sourceProtocolAddrRepeatCount</i> .
<i>ArpDecrement</i>	2	decrement by 1 for the count in <i>sourceProtocolAddrRepeatCount</i> .
<i>arpContinuousIncrement</i>	3	increment by 1 continuously.
<i>arpContinuousDecrement</i>	3	decrement by 1 continuously.

sourceProtocolAddrRepeatCount

Indicates the repeat count for the *sourceProtocolAddrMode* increment and decrement options. (*default = 0*)

COMMANDS

The **arp** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

arp cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **arp** command.

arp config option value

Modify the configuration options of the arp. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for arp.

arp decode capSlice [chassisID cardID portID]

Decodes a captured slice/frame into the arp variables. If not an arp frame, returns TCL_ERROR. May be used to determine if the captured frame is a valid arp frame. Specific errors are as follows:

- No connection to a chassis
- The captured frame is not an ARP frame

arp get chassisID cardID portID

Gets the current configuration of the arp frame for port with id *portID* on card *cardID*, chassis *chassisID*. from its hardware. Call this command before calling **arp cget option value** to get the value of the configuration option. Specific errors are as follows:

- No connection to a chassis
- Invalid port number

arp set chassisID cardID portID

Sets the configuration of the arp in IxHAL for port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **arp config option value** command. Specific errors are as follows:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

arp setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal
```

```
set host      woodstock
set username   user

# In this example, ports 1 and 2 of a card are directly connected
# together
# ArpDecrement 2 decrement by 1 for the count in
# sourceProtocolAddrRepeatCount.
# arpContinuousIncrement 3 increment by 1 continuously.
# arpContinuousDecrement 3 decrement by 1 continuously.
# Port 1 transmits an ARP request and looks at the response packet
# Port 2 uses its address table and protocol server to respond to
# the arp
# request

# Check if we're running on UNIX - connect to the TCL Server which
# must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

set chas      [chassis cget -id]
set card      2
set txPort    1
set rxPort    2

# Primary contents for port 2 arp table
set txPortMAC  {00 00 00 01 01 01}
set rxPortMAC  {00 00 00 01 01 02}
set txIP       {192.168.18.1}
set rxIP       {192.168.18.2}

# An extra entry for Vlan demonstration
set rxPortMAC2 {00 00 00 01 02 02}
set rxIP2      {192.168.28.2}

set portList [list [list $chas $card $txPort] [list $chas $card
$rxPort]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports to use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Set up Transmit Port
# Nothing special about the port setup
port setFactoryDefaults $chas $card $txPort
```

```

protocol setDefault
protocol config -name ipV4
protocol config -appName Arp
protocol config -ethernetType ethernetII

# Stream: 1 packet to broadcast address
stream setDefault
stream config -numFrames 1

# A 1 packet stream can't go at 100%
stream config -percentPacketRate 1
stream config -rateMode usePercentRate
stream config -sa $txPortMAC

# Broadcast arp request
stream config -da {ff ff ff ff ff ff}
stream config -dma stopStream

# Now set up the ARP request packet
arp setDefault
arp config -sourceProtocolAddr $txIP
arp config -destProtocolAddr $rxIP
arp config -operation arpRequest
arp config -sourceHardwareAddr $txPortMAC
arp config -destHardwareAddr {ff ff ff ff ff ff}

if { [arp set $ch $card $txPort] } {
    ixPuts "Error setting arp on port $ch $card $txPort"
    return 1
}

if { [stream set $ch $card $txPort 1] } {
    ixPuts "Error setting stream 1 on port $ch $card $txPort"
    return 1
}

# Set up Receive Port for automatic ARP response
# Nothing special about the port setup
port setFactoryDefaults $ch $card $rxPort

protocol setDefault
protocol config -ethernetType ethernetII

# Add an address table item for IP/MAC
ipAddressTable setDefault
ipAddressTableItem setDefault
ipAddressTableItem config -fromIpAddress $rxIP
ipAddressTableItem config -fromMacAddress $rxPortMAC
if { [ipAddressTable addItem] } {
    ixPuts "Error ipAddressTable addItem on $ch $card $rxPort"
    return 1
}

# Add another with Vlan set
ipAddressTableItem config -fromIpAddress $rxIP2
ipAddressTableItem config -fromMacAddress $rxPortMAC2
ipAddressTableItem config -enableVlan true
ipAddressTableItem config -vlanId 2
if { [ipAddressTable addItem] } {
    ixPuts "Error ipAddressTable addItem on $ch $card $rxPort"
    return 1
}
if { [ipAddressTable set $ch $card $rxPort] } {

```

```
    ixPuts "Error setting ipAddressTable on $chassis $card $rxPort"
    return 1
}

# Let the port respond to arp requests
protocolServer setDefault
protocolServer config -enableArpResponse true
if { [protocolServer set $chassis $card $rxPort] } {
    ixPuts "Error setting protocolServer on $chassis $card $rxPort"
    return 1
}

# Commit to hardware
if { [ixWritePortsToHardware portList] } {
    ixPuts "Error ixWritePortsToHardware"
    return 1
}

# Make sure link is up
after 3000

ixCheckLinkState portList

ixStartPortCapture $chassis $card $txPort
ixStartPortTransmit $chassis $card $txPort
after 1000
ixCheckPortTransmitDone $chassis $card $txPort
ixStopPortCapture $chassis $card $txPort

# Get the ARP response from the capture buffer
if { [captureBuffer get $chassis $card $txPort] } {
    ixPuts "Error getting captureBuffer on $chassis $card $txPort"
    return 1
}
if { [captureBuffer cget -numFrames] == 0} {
    ixPuts "No packets received"
} else {
    # and extract just the returned address
    if { [captureBuffer getframe 1] } {
        ixPuts "Error getframe"
        return 1
    }
    set data [captureBuffer cget -frame]
    set data [string range $data 84 94]
    ixPuts "ARP response (IP in hex): $data"
}

# Let go of the ports that we reserved
ixClearOwnership $portList

# Disconnect from the chassis we're using
if { [ixDisconnectFromChassis $host] } {
    ixPuts $::ixErrorInfo
    return 1
}

# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*ip, stream*

NAME - associationHeader

associationHeader-sets up Association Header over Fibre Channel.

SYNOPSIS

associationHeader *sub-command options*

DESCRIPTION

The Association Header is an optional header within the Data Field content. Its presence is indicated by bit 20 in the DF_CTL field, located in the Frame Header, being set to one. The Association Header is 32-bytes in size. The Association Header is used to identify a specific process or group of Processes within a node associated with an Exchange.

STANDARD OPTIONS

**originatorProcess
Associator**

It is the value used in the Association Header to identify an originator process or a group of processes within a node.

**responderProcess
Associator**

It is the value used in the Association Header to identify a responder process or a group of processes within a node.

validity

Denotes the validity of the Association Header.

EXAMPLES

See under *fibreChannel*.

SEE ALSO

fibreChannel.

NAME - atmFilter

atmFilter - set up capture filters based on ATM packet contents.

COMMANDS

The `atmFilter` command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

atmFilter **caet** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmFilter* command.

atmFilter config *option value*

Modify the configuration options of the *atmFilter*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *atmFilter*.

atmFilter get *chasID cardID portID vpi vci*

Gets the options associated with a particular VPI/VCI on a port. Specific errors are as follows:

- No entry for the VPI/VCI - port
- Port is not available
- ATM is not supported on this port

atmFilter set *chasID cardID portID vpi vci*

Sets the options associated with a particular VPI/VCI on a port. The port should be in the current reassembly list (see *atmReassembly*) before setting the filter. Specific errors are as follows:

- No connection to the chassis
- Invalid port - not available or in use
- Invalid VPI/VCI
- Invalid filter parameters
- ATM feature is not supported on this port

atmFilter setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set chassis 1
set card    42
set vpi     1
set vci    17

# Set port 1 to UDS1, match on Ox42 in the first 7 bits
atmFilter setDefault
atmFilter config -enable          true
atmFilter config -enableUds1      true
atmFilter config -comparisonData {42}
atmFilter config -comparisonMask {01}

if [atmFilter set $chassis $card 1 $vpi $vci] {
    ixPuts "Error in atmFilter set"
}

# Set port 2 to capture trigger, match on 8th bit on
atmFilter setDefault
atmFilter config -enable          true
atmFilter config -enableTrigger   true
atmFilter config -comparisonData {01}
atmFilter config -comparisonMask {FE}

if [atmFilter set $chassis $card 2 $vpi $vci] {
    ixPuts "Error in atmFilter set"
}
```

SEE ALSO

atmReassembly, *atmStat*

NAME - atmHeader

atmHeader - configure ATM header parameters.

SYNOPSIS	atmHeader <i>sub-command options</i>																		
DESCRIPTION	<p>The atmHeader command is used to configure the ATM header options which are used in streams configured with the <i>stream</i> command. Note that <i>stream</i> get must be called before this command's get sub-command.</p> <p>Note that different types of ATM encapsulation result in different length headers, as discussed in the following table.</p> <p style="text-align: center;">TABLE A-1. ATM Encapsulation Header Lengths</p> <table border="1"> <thead> <tr> <th>Encapsulation</th> <th>Header Length</th> </tr> </thead> <tbody> <tr> <td>LLC Snap Routed</td> <td>8</td> </tr> <tr> <td>LLC Bridged Ethernet / 802.3</td> <td>10</td> </tr> <tr> <td>LLC Bridged Ethernet / 802.3 No FCS</td> <td>10</td> </tr> <tr> <td>LLC Encapsulated PPP</td> <td>6</td> </tr> <tr> <td>VC Muxed PPP</td> <td>2</td> </tr> <tr> <td>VC Muxed Routed</td> <td>0</td> </tr> <tr> <td>VC Muxed Bridged Ethernet / 802.3</td> <td>2</td> </tr> <tr> <td>VC Muxed Bridged Ethernet / 802.3 No FCS</td> <td>2</td> </tr> </tbody> </table>	Encapsulation	Header Length	LLC Snap Routed	8	LLC Bridged Ethernet / 802.3	10	LLC Bridged Ethernet / 802.3 No FCS	10	LLC Encapsulated PPP	6	VC Muxed PPP	2	VC Muxed Routed	0	VC Muxed Bridged Ethernet / 802.3	2	VC Muxed Bridged Ethernet / 802.3 No FCS	2
Encapsulation	Header Length																		
LLC Snap Routed	8																		
LLC Bridged Ethernet / 802.3	10																		
LLC Bridged Ethernet / 802.3 No FCS	10																		
LLC Encapsulated PPP	6																		
VC Muxed PPP	2																		
VC Muxed Routed	0																		
VC Muxed Bridged Ethernet / 802.3	2																		
VC Muxed Bridged Ethernet / 802.3 No FCS	2																		

The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in the *dataIntegrity*, *filter*, *flexibleTimestamp*, *ip*, *ipV6Fragment*, *packetGroup*, *protocolOffset*, *qos*, *tableUdfColumn*, *tcp*, *udf*, and *udp* are with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

The framesize of an ATM packet is set by a combination of the *enableCpcsLength* and *cpcsLength* options in this command and the *framesize* option in the *stream* command. If *enableCpcsLength* is set to *true*, then the ATM frame's size is set from the *cpcsLength* value only. Otherwise, it is set from the *stream*'s *framesize* value and the *cpcsLength* value is calculated from that. Further, the *stream* *getQueue* command resets this command's *enableCpcsLength* option to *false*. It is important to correctly set the *stream*'s *framesize* value and this command's *enableCpcsLength* and *cpcsLength* options after each *stream* *getQueue* command and call *atmHeader set* before the next *stream* *setQueue* command.

STANDARD OPTIONS

aal5Error

May be used to insert a bad AAL5 CRC.

Option	Value	Usage
aal5NoError	0	(default) No error is inserted
aal5BadCrc	1	Inserts an AAL5 CRC error.

cellLossPriority

Sets the Cell Loss Priority, also abbreviated as CLPUsed to set the discard priority level of the cell. It indicates whether the cell should be discarded if it encounters extreme congestion as it moves through the network. Values of 0 and 1 are allowed, with 0 having a higher priority than 1. (default = 0)

cpcsLength

If *enableCpcsLength* is true, then this is used as the length of the CPCS PDU. (default = 28)

**enableAutoVpiVci
Selection true / false**

If *true*, the *vpi/vci* values are forced to 0 and 32. (default = false)

enableCL true / false

Indicates whether congestion has been experienced. (default = false)

**enableCpcsLength
true / false**

If *true*, the value of *cpcsLength* is used as the length of the CPCS PDU. The value of the framesize configured in the *stream* command is ignored. It is important to note that this value is always set to *false* by the *stream getQueue* command. (default = false)

encapsulation

The type of header encapsulation.

Option	Value	Usage
atmEncapsulationVccMuxIPV4Routed	101	
atmEncapsulationVccMuxBridgedEthernetFCS	102	
atmEncapsulationVccMuxBridgedEthernetNoFCS	103	
atmEncapsulationVccMuxIPV6Routed	104	
atmEncapsulationVccMuxMPLSRouted	105	
atmEncapsulationLLCRoutedCLIP	106	(default)
atmEncapsulationLLCBridgedEthernetFCS	107	
atmEncapsulationLLCBridgedEthernetNoFCS	108	
atmEncapsulationLLCPPPoA	109	
atmEncapsulationVccMuxPPPoA	110	
atmEncapsulationLCLNLPIDRouted	111	

genericFlowControl

Generic Flow Control for use in UNI mode device control signalling.
Uncontrolled equipment uses a setting of 0000. (default = 0)

header

Read-only. The 5-byte calculated header value.

hecErrors

Indicates the number of HEC errors to insert into the HEC byte. Values of 0 (no errors) through 8 (8 errors) are allowed. (default = 0)

vci

The virtual circuit identifier. (default = 32)

vpi	The virtual path identifier. (<i>default = 0</i>)
------------	---

COMMANDS	The <i>atmHeader</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

atmHeader cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmHeader* command.

atmHeader config option value

Modify the configuration options of the *atmHeader*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *atmHeader*.

atmHeader get chasID cardID portID

Gets the current configuration of the *atmHeader* for the port. Note that *stream* get must be called before this command's get sub-command. Call this command before calling *atmHeader cget option value* to get the value of the configuration option. Specific errors are as follows:

- No connection to a chassis
- Protocol data has not been configured for this port through a *stream set* or *protocol set*

atmHeader set chasID cardID portID

Sets the configuration of the *atmHeader* in IxHAL for the port by reading the configuration option values set by the **atmHeader config option value** command. Specific errors are as follows:

- No connection to a chassis
- Invalid port number
- Port unavailable or in use
- Configured parameters are not valid for this setting
- ATM is not supported on this port

atmHeader setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES	See examples in <i>stream</i> .
-----------------	---------------------------------

SEE ALSO	<i>atmPort, stream, streamQueue, streamQueueList</i>
-----------------	--

NAME - atmHeaderCounter

atmHeaderCounter - configure ATM counters for VPI and VCI values.

SYNOPSIS

atmHeaderCounter *sub-command options*

DESCRIPTION

The *atmHeaderCounter* command is used to configure the ATM configure a counter that can set the VPI or VCI value to:

- Fixed: a single value is used throughout.
- Counter: an incrementing counter is applied.
- Random: a masked set of bits are randomly set.
- Table: a table of values is repetitively used.

The selection of VPI or VCI is performed in the *set* and *get* sub-commands.

STANDARD OPTIONS

dataItemList

If the *type* option is set to *atmTableMode*, then this TCL list is used for the set of values. (*default* = {})

maskselect

If the *type* option is set to *atmRandom*, then this 16-bit mask indicates which bits are held constant. The constant values are indicated in the *maskvalue* option. (*default* = {00 00})

maskvalue

If the *type* option is set to *atmRandom*, then this 16-bit value indicates the values that the bits indicated in the *maskselect* option should have. (*default* = {00 00})

mode

If the *type* option is set to *atmCounter*, then this indicates what counter mode should be used.

Option	Value	Usage
<i>atmIncrement</i>	0	(<i>default</i>) Increment the VPI/VCI value for the number of times indicated in the <i>repeatCount</i> option by the value indicated in the <i>step</i> option. After the <i>repeatCount</i> is exhausted, the value from the <i>vpi</i> or <i>vci</i> option in the <i>atmHeader</i> command is used.
<i>atmContinuousIncrement</i>	1	Continuously increment the VPI/VCI value by the value indicated in the <i>step</i> option.
<i>atmDecrement</i>	2	Decrement the VPI/VCI value for the number of times indicated in the <i>repeatCount</i> option by the value indicated in the <i>step</i> option. After the <i>repeatCount</i> is exhausted, the value from the <i>vpi</i> or <i>vci</i> option in the <i>atmHeader</i> command is used.
<i>atmContinuousDecrement</i>	3	Continuously decrement the VPI/VCI value by the value indicated in the <i>step</i> option.

repeatCount

If the *type* option is set to *atmCounter* and the *mode* option is set to *atmIncrement* or *atmDecrement*, then this is the number of time to increment the VPI/VCI value before repeating from the start value. (*default* = 1)

step If the *type* option is set to *atmCounter*, then this is the value added/subtracted between successive values. (*default = 1*)

type The type of counter to use on the VPI/VCI.

Option	Value	Usage
<i>atmidle</i>	0	(<i>default</i>) The VPI/VCI has a fixed value set in the <i>atmHeader</i> command's <i>vpi</i> or <i>vci</i> option.
<i>atmCounter</i>	1	The VPI/VCI value increments or decrements for a fixed number of repetitions or continuously, as dictated by the <i>mode</i> and <i>repeatCount</i> options. The step size is in the <i>step</i> option. The starting value is set in the value set in the <i>atmHeader</i> command's <i>vpi</i> or <i>vci</i> option.
<i>atmRandom</i>	2	Selected bits of VPI/VCI value varies randomly. The mask of values that are fixed is in the <i>maskselect</i> option and their fixed values are in the <i>maskvalue</i> option.
<i>atmTableMode</i>	3	The VPI/VCI values are selected round-robin from the data table in the <i>dataItemList</i> option.

COMMANDS

The *atmHeaderCounter* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

atmHeaderCounter **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmHeaderCounter* command.

atmHeaderCounter **config** *option value*

Modify the configuration options of the *atmHeaderCounter*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *atmHeaderCounter*.

atmHeaderCounter **get** *type*

Gets the current configuration of the *atmHeaderCounter* for the VPI/VCI type indicated in the *type* argument. The choices are:

Option	Value	Usage
<i>atmVpi</i>	0	(<i>default</i>) The VPI value.
<i>atmVci</i>	1	The VCI value.

Call this command before calling *atmHeaderCounter cget option value* to get the value of the configuration option. Specific errors are:

- Invalid *type*

atmHeaderCounter **set** *type*

Sets the current configuration of the *atmHeaderCounter* for the VPI/VCI type indicated in the *type* argument. The choices are:

Option	Value	Usage
<i>atmVpi</i>	0	(<i>default</i>) The VPI value.
<i>atmVci</i>	1	The VCI value.

Specific errors are:

- Invalid *type*
- Invalid parameter settings

atmHeaderCounter **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples in *stream*.

SEE ALSO

atmPort, *stream*, *streamQueue*, *streamQueueList*

NAME - atmOam

atmOam - configure ATM OAM messages to be transmitted

SYNOPSIS

atmOam *sub-command options*

DESCRIPTION

The **atmOam** command is used to configure multiple ATM OAM messages to be transmitted on an ATM port. The basic parameters for all OAM messages are configured in the options of this command. Additional parameters that are particular to a specific OAM message are taken from the following additional commands: *atmOamActDeact*, *atmOamAis*, *atmOamFaultManagementCC*, *atmOamFaultManagementLB* or *atmOamRdi*.

Once configured, the OAM message for a VPI/VCI pair is added to the list associated with this command with the *add* sub-command. Transmission of the OAM messages is initiated with the *start* sub-command and stopped with the *stop* sub-command.

Trace information, if enabled with the *enableTrace* option is retrieved using the *atmOamTrace* command.

STANDARD OPTIONS

cellFlowType

The cell flow type for the OAM message.

Option	Value	Usage
<i>atmOamF4</i>	0	F4 flow.
<i>atmOamF5</i>	1	(default) F5 flow.

enableCC *true / false*

If *true*, enables continuous checking. (*default = false*)

enableLB *true / false*

If *true*, enables loopback. (*default = false*)

enableTrace *true / false*

If *true*, trace messages per registered VPI/VCI pair is enabled. (*default = false*)

enableTx *true / false*

If *true*, the current OAM message is enabled for transmission. (*default = true*)

endPointsType

The endpoint type.

Option	Value	Usage
<i>atmOamEndToEnd</i>	0	(default) End to end.
<i>atmOamSegment</i>	1	Segment.

functionType

The OAM function to be performed.

Option	Value	Usage
<i>atmOamAis</i>	0	(default) AIS. Additional message options are obtained from the <i>atmOamAis</i> command.
<i>atmOamRdi</i>	1	RDI. Additional message options are obtained from the <i>atmOamRdi</i> command.
<i>atmOamFaultMgmtCC</i>	2	Fault Management CC. Additional message options are obtained from the <i>atmOamFaultManagementCC</i> command.

Option	Value	Usage
<i>atmOamFaultMgmtLB</i>	3	Fault Management LB. Additional message options are obtained from the <i>atmOamFaultManagementLB</i> command.
<i>atmOamActDeactCC</i>	4	Activate-Deactivate. Additional message options are obtained from the <i>atmOamActDeact</i> command.

vci

Read-only. The VCI for the registered OAM cell for list entries retrieved by one of the *get* sub-commands. The VCI value is set in the add sub-command. (*default = 0*)

vpi

Read-only. The VPI for the registered OAM cell for list entries retrieved by one of the *get* sub-commands. The VPI value is set in the add sub-command (*default = 0*)

COMMANDS

The *atmOam* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

atmOam add vpi vci

Adds the VPI/VCI pair to the OAM list for the indicated port. Based on the OAM type specified in *functionType*, additional message options are taken from one of the *atmOamActDeact*, *atmOamAis*, *atmOamFaultManagementCC*, *atmOamFaultManagementLB* or *atmOamRdi* commands. Specific errors are:

- *select* has not been called
- The port is in use by another user
- ATM is not supported on this port
- Invalid port
- The maximum number of ATM OAM entries has been exceeded.
- The VPI/VCI is already in the list.

atmOam cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmOam* command.

atmOam config option value

Modify the configuration options of the *atmOam*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *atmOam*.

atmOam del vpi vci

Removes the VPI/VCI pair from the OAM list for the indicated port. Specific errors are:

- *select* has not been called
- The port is in use by another user
- ATM is not supported on this port
- The VPI/VCI pair is not in the list

atmOam getFirstPair

Accesses the first VPI/VCI pair in the list, whose values can be found in the STANDARD OPTIONS for this command and the *functionType* specific commands. Specific errors are:

- *select* has not been called
- No connection to the chassis
- ATM is not supported on this port
- Invalid port
- No pairs in the list.

atmOam **getNextPair**

Accesses the next VPI/VCI pair in the list, whose values can be found in the *STANDARD OPTIONS* for this command and the *functionType* specific commands. Specific errors are:

- *select* has not been called
- *getFirstPair* has not been called
- ATM is not supported on this port
- Invalid port
- No more pairs in the list.

atmOam **removeAll**

Removes all VPI/VCI pairs from the OAM list for the indicated port. Specific errors are:

- *select* has not been called
- ATM is not supported on this port
- The port is in use by another user
- Invalid port

atmOam **select chasID cardID portID**

Accesses the indicated port. All other sub-commands uses this port. Specific errors are:

- No connection to the chassis
- Invalid port specified
- Port is not available
- ATM OAM is not an available feature for the port

atmOam **setDefault**

Sets to IxTclHal default values for all configuration options.

atmOam **start chasID cardID portID**

Starts transmission of the ATM OAM messages on the indicated port. Specific errors are:

- No connection to the chassis
- ATM is not supported on this port
- The port is in use by another user
- Invalid port

atmOam **stop chasID cardID portID**

Stops transmission of the ATM OAM messages on the indicated port. Specific errors are:

- No connection to the chassis
- ATM is not supported on this port
- The port is in use by another user

- Invalid port
- Transmission has not been started

EXAMPLES

```

package req IxTclHal

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chassisId [ixGetChassisID $host]

set cardId 26
set portId 1
set portList      [list [list $chassisId $cardId $portId]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

if [port isValidFeature $chassisId $cardId $portId
$::portFeatureAtmPos] {
    port setFactoryDefaults $chassisId $cardId $portId
    port config -portMode portAtmMode
    if {[port set $chassisId $cardId $portId] } {
        errorMsg "Error setting port on port $chassisId $cardId
$portId"
        return 1
    }

    stat config -enableAtmOamStats $::true
    if {[stat set $chassisId $cardId $portId] } {
        errorMsg "Error setting stats on port $chassisId $cardId
$portId"
        return 1
    }

    atmOamTrace config -maxNumTrace 50
    if {[atmOamTrace set $chassisId $cardId $portId] } {

```

```

        errorMsg "Error setting oam trace for port $chassId $cardId
$portId"
        set retCode "FAIL"
    }
} else {
    errorMsg "Port $chassId:$cardId:$portId is not an ATM port"
    return 1
}

ixWritePortsToHardware portList

if {[atmOam select $chassId $cardId $portId]} {
    errorMsg "Error selecting atmOam on $chassId $cardId $portId."
    return 1
}

atmOam removeAll

atmOam setDefault

atmOam config -functionType atmOamFaultMgmtLB
atmOam config -cellFlowType atmOamF5
atmOam config -endPointType atmOamSegment
atmOam config -enableLB true
atmOam config -enableCC true
atmOam config -enableTrace true
atmOam config -enableTx true

atmOamFaultManagementLB config -enableTxContinuous false
atmOamFaultManagementLB config -txCount 5
atmOamFaultManagementLB config -loopbackIndication atmOamReply
atmOamFaultManagementLB config -correlationTag "11 11 11 11"
atmOamFaultManagementLB config -loopbackLocationId "55 55 55 55 55 55 55 55 55 55 55 55"
55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55

if [atmOam add 31 32] {
    ixPuts $::ixErrorInfo
    return 1
}

atmOam setDefault

atmOam config -functionType atmOamAis
atmOam config -cellFlowType atmOamF4
atmOam config -endPointType atmOamEndToEnd
atmOam config -enableLB true
atmOam config -enableCC true
atmOam config -enableTrace true
atmOam config -enableTx true

atmOamAis config -enableTxContinuous false
atmOamAis config -txCount 6

if [atmOam add 33 4] {
    ixPuts $::ixErrorInfo
    return 1
}

atmOam setDefault

atmOam config -functionType atmOamRdi
atmOam config -cellFlowType atmOamF4
atmOam config -endPointType atmOamSegment

```

```

atmOam config -enableLB true
atmOam config -enableCC false
atmOam config -enableTrace true
atmOam config -enableTx true

atmOamRdi config -enableTxContinuous false
atmOamRdi config -txCount 8

if [atmOam add 14 4] {
    ixPuts $::ixErrorInfo
    return 1
}

atmOam setDefault

atmOam config -functionType atmOamFaultMgmtCC
atmOam config -cellFlowType atmOamF4
atmOam config -endPointType atmOamEndToEnd
atmOam config -enableLB false
atmOam config -enableCC true
atmOam config -enableTrace true
atmOam config -enableTx true

atmOamFaultManagementCC config -enableTxContinuous false
atmOamFaultManagementCC config -txCount 4

if [atmOam add 37 4] {
    ixPuts $::ixErrorInfo
    return 1
}

atmOam setDefault

atmOam config -functionType atmOamActDeactCC
atmOam config -cellFlowType atmOamF5
atmOam config -endPointType atmOamSegment
atmOam config -enableLB true
atmOam config -enableCC false
atmOam config -enableTrace false
atmOam config -enableTx true

atmOamActDeact config -enableTxContinuous false
atmOamActDeact config -txCount 11
atmOamActDeact config -messageID atmOamDeactivate
atmOamActDeact config -action atmOamAB
atmOamActDeact config -correlationTag 0x11

if [atmOam add 1 2] {
    ixPuts $::ixErrorInfo
    return 1
}

ixWriteConfigToHardware portList

set numTrace 10
set maxNumTrace 50
set numTracePerDirection [expr $numTrace/2]

set oamTxFMLB 5
set oamTxRDI 8

ixCheckLinkState portList

```

```

ixClearStats portList
after 1000

if {[atmOam start $chassId $cardId $portId]} {
   errorMsg "Error starting oam transmit on port $chassId $cardId
$portId"
    return 1
}
after 3000

atmOamTrace setDefault

while {[atmOamTrace cget -numTrace] != $maxNumTrace} {
    if {[atmOamTrace get $chassId $cardId $portId]} {
        errorMsg "Error getting oam trace for port $chassId $cardId
$portId"
        return 1
    }
    after 1000
}

if {[atmOamTrace getFirst]} {
    errorMsg "Error getting first trace for port $chassId $cardId
$portId"
    return 1
}

stat get allStats $chassId $cardId $portId
set oatmTxFMLB [stat cget -atmOamTxFaultMgmtLB]
set oatmTxRDI [stat cget -atmOamTxFaultMgmtRDI]

# Now use the hight level APIs
if {[ixStartAtmOamTransmit portList]} {
    errorMsg "Error ixStartAtmOamTransmit"
    return 1
}

if {[ixStopAtmOamTransmit portList]} {
    errorMsg "Error ixStopAtmOamTransmit"
    return 1
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
return 0

```

SEE ALSO

*atmOamActDeact, atmOamAis, atmOamFaultManagementCC,
atmOamFaultManagementLB, atmOamRdi, atmOamTrace, atmPort*

NAME - atmOamActDeact

atmOamActDeact - configure an ATM OAM activation/deactivation message

atmOamActDeact **cget *option***

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmOamActDeact* command.

atmOamActDeact **config *option value***

Modify the configuration options of the *atmOamActDeact*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *atmOamActDeact*.

atmOamActDeact **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *atmOam*.

SEE ALSO

atmOam, *atmOamAis*, *atmOamFaultManagementCC*,
atmOamFaultManagementLB, *atmOamRdi*, *atmOamTrace*, *atmPort*

NAME - atmOamAis

atmOamAis - configure an ATM OAM AIS message

SYNOPSIS	atmOamAis <i>sub-command options</i>
DESCRIPTION	The atmOamAis command holds command specific options for the AIS message.
STANDARD OPTIONS	
enableTxContinuous <i>true / false</i>	If <i>true</i> , the message is transmitted continuously. (<i>default = true</i>)
reserved	<i>Read-only</i> . The value of the reserved field, which may not be modified.
txCount	If <i>enableTxContinuous</i> is <i>false</i> , the count of the number of times that the message is transmitted. (<i>default = 0</i>)
COMMANDS	
	The <i>atmOamAis</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
atmOamAis cget <i>option</i>	
	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the <i>atmOamAis</i> command.
atmOamAis config <i>option value</i>	
	Modify the configuration options of the <i>atmOamAis</i> . If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for <i>atmOamAis</i> .
atmOamAis setDefault	
	Sets to IxTclHal default values for all configuration options.
EXAMPLES	
	See examples under <i>atmOam</i> .
SEE ALSO	
	<i>atmOam, atmOamActDeact, atmOamFaultManagementCC, atmOamFaultManagementLB, atmOamRdi, atmOamTrace, atmPort</i>

NAME - atmOamFaultManagementCC

atmOamFaultManagementCC - configure an ATM OAM Fault Management CC message

SYNOPSIS	atmOamFaultManagementCC <i>sub-command options</i>
-----------------	--

DESCRIPTION	The atmOamFaultManagementCC command holds command specific options for the Fault Management CC message.
--------------------	--

STANDARD OPTIONS

enableTxContinuous <i>true / false</i>	If <i>true</i> , the message is transmitted continuously. (<i>default = true</i>)
reserved	<i>Read-only</i> . The value of the reserved field, which may not be modified.
txCount	If <i>enableTxContinuous</i> is <i>false</i> , the count of the number of times that the message is transmitted. (<i>default = 0</i>)

COMMANDS	The <i>atmOamFaultManagementCC</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

atmOamFaultManagementCC **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmOamFaultManagementCC* command.

atmOamFaultManagementCC **config** *option value*

Modify the configuration options of the *atmOamFaultManagementCC*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *atmOamFaultManagementCC*.

atmOamFaultManagementCC **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES	See examples under atmOam .
-----------------	---

SEE ALSO	atmOam , atmOamActDeact , atmOamAis , atmOamFaultManagementLB , atmOamRdi , atmOamTrace , atmPort
-----------------	---

NAME - atmOamFaultManagementLB

atmOamFaultManagementLB - configure an ATM OAM Fault Management LB message

SYNOPSIS

atmOamFaultManagementLB *sub-command options*

DESCRIPTION

The **atmOamFaultManagementLB** command holds command specific options for the Fault Management LB message.

STANDARD OPTIONS

correlationTag

The correlation tag. (*default = "00 00 00 00 00"*)

enableTxContinuous *true / false*

If *true*, the message is transmitted continuously. (*default = true*)

loopbackIndication

The loopback indication.

Option	Value	Usage
<i>atmOamReply</i>	0	(<i>default</i>)
<i>atmOamRequest</i>	1	

loopbackIndicationId

The loopback indication ID. (*default = “FF FF FF”*)

reserved

Read-only. The value of the reserved field, which may not be modified.

sourceLocationId

The source location ID. (*default = “00 00”*)

txCount

If *enableTxContinuous* is *false*, the count of the number of times that the message is transmitted. (*default = 0*)

COMMANDS

The *atmOamFaultManagementLB* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

atmOamFaultManagementLB **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmOamFaultManagementLB* command.

atmOamFaultManagementLB **config** *option value*

Modify the configuration options of the *atmOamFaultManagementLB*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *atmOamFaultManagementLB*.

atmOamFaultManagementLB **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *atmOam*.

SEE ALSO

atmOam, atmOamActDeact, atmOamAis, atmOamFaultManagementCC, atmOamRdi, atmOamTrace, atmPort

NAME - atmOamRdi

atmOamRdi - configure an ATM OAM RDI message

SYNOPSIS	atmOamRdi <i>sub-command options</i>
DESCRIPTION	The atmOamRdi command holds command specific options for the RDI message.
STANDARD OPTIONS	
defectLocation	The defect location. (<i>default</i> = “00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00”)
defectType	<i>Read-only</i> . The defect type, currently set to 0x6A.
enableTxContinuous <i>true / false</i>	If <i>true</i> , the message is transmitted continuously. (<i>default</i> = <i>true</i>)
reserved	<i>Read-only</i> . The value of the reserved field, which may not be modified.
txCount	If <i>enableTxContinuous</i> is <i>false</i> , the count of the number of times that the message is transmitted. (<i>default</i> = 0)
COMMANDS	
	The <i>atmOamRdi</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
atmOamRdi cget <i>option</i>	
	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the <i>atmOamRdi</i> command.
atmOamRdi config <i>option value</i>	
	Modify the configuration options of the <i>atmOamRdi</i> . If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for <i>atmOamRdi</i> .
atmOamRdi setDefault	
	Sets to IxTclHal default values for all configuration options.
EXAMPLES	
	See examples under <i>atmOam</i> .
SEE ALSO	
	<i>atmOam, atmOamActDeact, atmOamAis, atmOamFaultManagementCC, atmOamFaultManagementLB, atmOamTrace, atmPort</i>

NAME - atmOamTrace

atmOamTrace - configure ATM OAM messages to be transmitted

SYNOPSIS

atmOamTrace *sub-command options*

DESCRIPTION

The **atmOamTrace** command is used to retrieve ATM OAM messages. These are collected for any OAM message in which the *enableTrace* option was set to true when *atmOam add* was called.

Messages are collected into a circular buffer of *maxNumTrace* messages in size. Newest entries replace oldest entries as necessary. The *get chassis card port* sub-command is used to retrieve all of the message. The other *get* commands are used to look at particular entries.

STANDARD OPTIONS

cellInformation *Read-only*. Cell information for the trace cell.

functionType *Read-only*. The OAM function type.

Option	Value	Usage
<i>atmOamAis</i>	0	AIS.
<i>atmOamRdi</i>	1	RDI.
<i>atmOamFaultMgmtCC</i>	2	Fault Management CC.
<i>atmOamFaultMgmtLB</i>	3	Fault Management LB.
<i>atmOamActDeactCC</i>	4	Activate - Deactivate.

maxNumTrace The maximum number of traces to be stored in the in-memory buffer. The buffer is used in a circular manner, with the most recent traces overwriting the oldest entries. (*default = 256*)

numTrace *Read-only*. The number of trace messages currently in the list.

timeStamp *Read-only*. The timestamp of the trace message, in the format:

YYYY/MM/DD HH: MM: SS.SSS

traceString *Read-only*. The trace message as a complete string.

txRxType *Read-only*. An indication of whether the trace is from a transmission or reception.

Option	Value	Usage
<i>atmOamTraceTx</i>	0	Transmit
<i>atmOamTraceRx</i>	1	Receive

vci *Read-only*. The VCI value from the trace message.

vpi *Read-only*. The VPI value from the trace message.

COMMANDS

The *atmOamTrace* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

atmOamTrace **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmOamTrace* command.

atmOamTrace **config** *option value*

Modify the configuration options of the *atmOamTrace*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *atmOamTrace*.

atmOamTrace **clear**

Removes all trace messages from the list.

atmOamTrace **get** *chassisID cardID portID*

atmOamTrace **get** *traceIndex*

In the first form, the trace list is retrieved. The first trace message is unpacked into the options of this command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- ATM is not a supported feature on this port
- There is no trace information to get

In the second form, the trace message at *traceIndex* is accessed and unpacked into the options of this command. The first message has a *traceIndex* of 0. A call to *getNext* accesses the message following *traceIndex*. Specific errors are:

- No trace message exists for the indicated *traceIndex*.

atmOamTrace **getFirst**

Accesses the first trace message in the list, whose values can be found in the STANDARD OPTIONS for this command. Specific errors are:

- There are no trace messages.

atmOamTrace **getNext**

Accesses the next trace message in the list, whose values can be found in the STANDARD OPTIONS for this command. Specific errors are:

- There are no more trace messages.

atmOamTrace **set** *chassisID cardID portID*

Sets the configuration of the *atmOamTrace* in IxHAL for the port indicated. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- ATM is not a supported feature on this port

atmOamTrace **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *atmOam*.

SEE ALSO

atmOam, atmOamActDeact, atmOamAis, atmOamFaultManagementCC, atmOamFaultManagementLB, atmOamRdi, atmPort

NAME - atmPort

atmPort - configure ATM port parameters.

transmitStatMode

This setting controls statistics are collected for the ATM port.

Option	Value	Usage
<i>atmPerVPIVCIStrats</i>	0	(default) Collect statistics for the whole port on a VCI/VPI basis.
<i>atmDecodeCell</i>	1	Collect statistics for the port on a stream basis.

COMMANDS

The *atmPort* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

atmPort cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmPort* command.

atmPort config option value

Modify the configuration options of the *atmPort*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *atmPort*.

atmPort get chasID cardID portID

Gets the current configuration of the *atmPort* for the port. Call this command before calling *atmPort cget option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- ATM is not a supported feature on this port

atmPort set chasID cardID portID

Sets the configuration of the *atmPort* in IxHAL for port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **atmPort config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting
- ATM is not a supported feature on this port

atmPort setDefault

Sets to IxTelHal default values for all configuration options.

EXAMPLES**SEE ALSO**

atmHeader, *atmOam*, *stream*, *streamQueue*, *streamQueueList*

NAME - atmReassembly

atmReassembly - configure ATM port to reassemble particular VPI/VCI streams

SYNOPSIS

atmReassembly *sub-command options*

DESCRIPTION

The **atmReassembly** command is used to configure an ATM port to reassemble received data for particular VPI/VCIs. This is necessary if a receive port is to be used in an *atmStat* receive list or in *atmFilter*. Note that these commands automatically calls this command for the port, if it is not in the reassembly list. Except for receive ports using other than default encapsulation (*atmEncapsulationLLCRoutedCLIP*) in packet group mode, the *add* sub-command need never be called; the *del* and *removeAll* commands proves useful when changing a list.

STANDARD OPTIONS

enableIpTcpUdp

Checksum true / false

Enables the collection of TCP and UDP checksum statistics for packets that match this VCI/VPI. (*default = 1*)

enableIpQos

true / false

Enables the collection of QoS statistics for packets that match this VCI/VPI. (*default = 1*)

encapsulation

The decode encapsulation to be used on received data when the port is in packet group mode. This is the only means by which the encapsulation may be set; calls from *atmStat* and *atmFilter* uses the default (*atmEncapsulationLLCRoutedCLIP*).

Option	Value	Usage
atmEncapsulationVccMuxIPv4Routed	101	
atmEncapsulationVccMuxBridgedEthernetFCS	102	
atmEncapsulationVccMuxBridgedEthernetNoFCS	103	
atmEncapsulationVccMuxIPv6Routed	104	
atmEncapsulationVccMuxMPLSRouted	105	
atmEncapsulationLLCRoutedCLIP	106	(<i>default</i>)
atmEncapsulationLLCBridgedEthernetFCS	107	
atmEncapsulationLLCBridgedEthernetNoFCS	108	
atmEncapsulationLLC PPPoA	109	
atmEncapsulationVccMuxPPPoA	110	

vci

Read-only. The current VCI.

vpi

Read-only. The current VPI.

COMMANDS

The *atmReassembly* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

atmReassembly add *chasID cardID portID vpi vci*

Adds the *vpi/vci* pair to the reassembly list for the indicated port. Specific errors are:

- The port is in use by another user
- ATM is not supported on this port
- Invalid port
- Invalid vci/vpi pair
- Item already in the list

atmReassembly cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmReassembly* command.

atmReassembly del *chasID cardID portID vpi vci*

Removes the *vpi/vci* pair from the reassembly list for the indicated port. Specific errors are:

- The port is in use by another user
- ATM is not supported on this port
- Item is not in the list

atmReassembly getFirstPair *chasID cardID portID*

Accesses the first VPI/VCI pair in the list, whose values can be found in the *STANDARD OPTIONS* for this command. Specific errors are:

- No connection to the chassis
- ATM is not supported on this port
- Invalid port
- No pairs in the list.

atmReassembly getNextPair *chasID cardID portID*

Accesses the next VPI/VCI pair in the list, whose values can be found in the *STANDARD OPTIONS* for this command. Specific errors are:

- No connection to the chassis
- ATM is not supported on this port
- Invalid port
- No more pairs in the list.

atmReassembly removeAll *chasID cardID portID*

Removes all *vpi/vci* pairs from the reassembly list for the indicated port. Specific errors are:

- No connection to the chassis
- ATM is not supported on this port
- The port is in use by another user
- Invalid port

atmReassembly setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set chassis 1
set card    42
set vpi     1
set vci    17

if [atmReassembly removeAll $chassis $card 1] {
    ixPuts "Error in atmReassembly removeAll"
}

if [atmReassembly removeAll $chassis $card 2] {
    ixPuts "Error in atmReassembly removeAll"
}

if [atmReassembly add $chassis $card 1 $vpi $vci] {
    ixPuts "Error in atmReassembly add"
}

if [atmReassembly add $chassis $card 2 $vpi $vci] {
    ixPuts "Error in atmReassembly add"
}

if [atmReassembly del $chassis $card 1 $vpi $vci] {
    ixPuts "Error in atmReassembly del"
}

if [atmReassembly getFirstPair $chassis $card 1] {
    ixPuts "No pairs in the list"
}

if [atmReassembly getNextPair $chassis $card 1] {
    ixPuts "No more pairs in the list"
}
```

SEE ALSO

atmFilter, *atmStat*

NAME - atmStat

atmStat - access VPI/VCI specific statistics.

SYNOPSIS	atmStat <i>sub-command options</i>
DESCRIPTION	The atmStat command is used to access statistics for particular VPI/VCI streams. VPI/VCIs for particular ports are added to a receive or transmit list with the <i>addRx</i> and <i>addTx</i> sub-commands. The statistics for all ports and VPI/VCIs in the lists is retrieved from the ports with the <i>get</i> sub-command. Individual statistics or rate statistics are accessed through the use of the <i>getStat</i> and <i>getRate</i> commands. The statistics are available in the STANDARD OPTIONS.
STANDARD OPTIONS	
rxAal5CrcErrors	<i>Read-only. 64-bit value.</i> The number/rate of received CRC errors.
rxAal5Frames	<i>Read-only. 64-bit value.</i> The number/rate of received CRC errors.
rxAal5LengthErrors	<i>Read-only. 64-bit value.</i> The number/rate of received length errors.
rxAal5TimeoutErrors	<i>Read-only. 64-bit value.</i> The number/rate of received timeout errors.
rxAtmCells	<i>Read-only. 64-bit value.</i> The number/rate of received ATM cells.
txAal5Bytes	<i>Read-only. 64-bit value.</i> The number/rate of transmitted AAL bytes.
txAal5Frames	<i>Read-only. 64-bit value.</i> The number/rate of transmitted AAL frames.
txAal5ScheduledBytes	<i>Read-only. 64-bit value.</i> The number/rate of transmitted AAL bytes, not including idle cells.
txAal5ScheduleFrames	<i>Read-only. 64-bit value.</i> The number/rate of transmitted AAL frames, not including idle cells.
txAtmCells	<i>Read-only. 64-bit value.</i> The number/rate of received ATM cells.
vci	<i>Read-only.</i> The current VCI.
vpi	<i>Read-only.</i> The current VPI.
COMMANDS	
	The <i>atmStat</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

atmStat **addRx** *chasID cardID portID vpi vci*

Adds the VPI/VCI for the indicated port to the receive list. If the 5-tuple is not in the current reassembly list (see *atmReassembly*), then it is automatically added to the list with the default encapsulation (*atmLlcSnapRoutedProtocol*). The encapsulation is only used for ports in packet group mode to correctly identify the packet group ID. Receive ports with other than default encapsulation should

first be added by [atmReassembly](#) and then added with this sub-command. Specific errors include:

- Invalid port
- VPI/VCI already exists
- The port is in use by another user
- No chassis connection
- ATM is not supported on this port
- The maximum number of Rx stats has been exceeded

atmStat **addTx** *chasID cardID portID vpi vci*

Adds the VPI/VCI for the indicated port to the transmit list. Specific errors include:

- Invalid port
- VPI/VCI already exists
- The port is in use by another user
- No chassis connection
- ATM is not supported on this port
- The maximum number of Tx stats has been exceeded

atmStat **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *atmStat* command.

atmStat **config** *option value*

Modify the configuration options of the *atmStat*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *atmStat*.

atmStat **delRx** *chasID cardID portID vpi vci*

Removes the VPI/VCI for the indicated port from the receive list. The 5-tuple is **not** removed from the current reassembly list (see [atmReassembly](#)). Specific errors include:

- The port is in use by another user
- No chassis connection
- ATM is not supported on this port
- Invalid port
- Invalid VPI/VCI
- The item is not in the receive list

atmStat **delTx** *chasID cardID portID vpi vci*

Removes the VPI/VCI for the indicated port from the transmit list. Specific errors include:

- The port is in use by another user
- No chassis connection
- ATM is not supported on this port
- Invalid port
- Invalid VPI/VCI
- The item is not in the transmit list

atmStat get

Gets the statistics for all of the VCI/VPIs added to the command using *addTx* and *addRx*. Specific errors are:

- Invalid port
- The port is in use by another user
- No connection to a chassis
- ATM is not supported on this port

atmStat getFirstRxPair *chasID cardID portID*

Accesses the first VPI/VCI pair in the receive list, whose values can be found in the *STANDARD OPTIONS* for this command. Specific errors are:

- The port is in use by another user
- No chassis connection
- ATM is not supported on this port
- Invalid port
- No pairs in the list.

atmStat getFirstTxPair *chasID cardID portID*

Accesses the first VPI/VCI pair in the transmit list, whose values can be found in the *STANDARD OPTIONS* for this command. Specific errors are:

- The port is in use by another user
- No chassis connection
- ATM is not supported on this port
- Invalid port
- No pairs in the list.

atmStat getNextRxPair *chasID cardID portID*

Accesses the next VPI/VCI pair in the receive list, whose values can be found in the *STANDARD OPTIONS* for this command. Specific errors are:

- The port is in use by another user
- No chassis connection
- ATM is not supported on this port
- Invalid port
- No more pairs in the list.

atmStat getNextTxPair *chasID cardID portID*

Accesses the next VPI/VCI pair in the transmit list, whose values can be found in the *STANDARD OPTIONS* for this command. Specific errors are:

- The port is in use by another user
- No chassis connection
- ATM is not supported on this port
- Invalid port
- No more pairs in the list.

atmStat getRate *chasID cardID portID vpi vci*

Makes all of the rate statistics for the particular VPI/VCI on the port available through the *STANDARD OPTIONS* of this command. Specific errors are:

- The port is in use by another user

- No chassis connection
- The VPI/VCI pair is not included in either the receive or transmit list
- ATM is not supported on this port

atmStat getStat *chasID cardID portID vpi vci*

Makes all of the statistics for the particular VPI/VCI on the port available through the STANDARD OPTIONS of this command. Specific errors are:

- The port is in use by another user
- No chassis connection
- The VPI/VCI pair is not included in either the receive or transmit list
- ATM is not supported on this port

atmStat removeAllRx *chasID cardID portID*

Removes all the VPI/VCI for the indicated port from the receive list. Specific errors include:

- The port is in use by another user
- No chassis connection
- ATM is not supported on this port
- Invalid port

atmStat removeAllTx *chasID cardID portID*

Removes all the VPI/VCI for the indicated port from the transmit list. Specific errors include:

- The port is in use by another user
- No chassis connection
- ATM is not supported on this port
- Invalid port

atmStat setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set chassis 1
set card    42
set vpi     1
set vci    17

# Remove all TX and RX ports for port
atmStat removeAllRx $chassis $card 1
atmStat removeAllTx $chassis $card 1
atmStat removeAllRx $chassis $card 2
atmStat removeAllTx $chassis $card 2

# Add both ports to both lists
if [atmStat addRx $chassis $card 1 $vpi $vci] {
    ixPuts "Error in atmStat addRx"
}
if [atmStat addRx $chassis $card 2 $vpi $vci] {
    ixPuts "Error in atmStat addRx"
}
if [atmStat addTx $chassis $card 1 $vpi $vci] {
```

```
        ixPuts "Error in atmStat addTx"
    }
    if [atmStat addTx $chassis $card 2 $vpi $vci] {
        ixPuts "Error in atmStat addTx"
    }

    # .... run some traffic ...

    # Get the statistics data for all the ports and VPI/VCI
    if [atmStat get] {
        ixPuts "Error in atmStat get"
    }
    after 2000

    if [atmStat getStat $chassis $card 1 $vpi $vci] {
        ixPuts "Error in atmStat getStat"
    }

    ixPuts "Port 1: [atmStat get -txAtmCells] cells transmitted, \
[atmStat get -rxAtmCells] received"

    if [atmStat getRate $chassis $card 2 $vpi $vci] {
        ixPuts "Error in atmStat getStat"
    }

    ixPuts "Port 2: [atmStat get -txAtmCells] cells transmitted/sec, \
[atmStat get -rxAtmCells] received/sec"
```

SEE ALSO

atmFilter, *atmReassembly*

NAME - autoDetectInstrumentation

autoDetectInstrumentation - configure auto-detection port parameters.

SYNOPSIS	autoDectectInstrumentation <i>sub-command options</i>
DESCRIPTION	The autoDectectInstrumentation command is used to configure the auto detection receive mode port options.
STANDARD OPTIONS	
enableSignatureMask <i>true/false</i>	Enables a mask of for the auto detect signature (<i>default = false</i>).
enableTxAutomatic Instrumentation	Transmit side only. Enables/disables the transmit options necessary to generate auto-detect instrumentation streams.
signature	Sets the auto detect signature (<i>default = 87 73 67 49 42 87 11 80 08 71 18 05</i>)
signatureMask	Sets the signature mask.
startOfScan	Sets an offset for where in the packet the auto detect should start looking for the signature (in bytes).
enableMisdirected PacketMask <i>true/false</i>	Enables/disables misdirected packet detection (<i>default = false</i>).
misdirectedPacketMask	Sets the misdirected packet mask (<i>Default = '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'</i>)
enablePRBS <i>true/false</i>	Enables the stream to transmit PRBS packets. (<i>default = false</i>) PRBS is enabled on a per-port basis for capture of PRBS packets. Note: This parameter is not supported by all load modules.
COMMANDS	
	The <i>autoDetectInstrumentation</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
autoDetectInstrumentation cget <i>option</i>	
	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the autoDetectInstrumentation command.
autoDetectInstrumentation getCircuitTx <i>chasID cardID portID [circuitID] streamID</i>	
	Gets the current configuration of the stream with id <i>streamID</i> in the circuit with <i>circuitID</i> on port <i>portID</i> , card <i>cardID</i> , chassis <i>chasID</i> from its hardware.

autoDetectInstrumentation **getQueueTx *chasID cardID portID [queueID] [streamID] [sequenceType]***

Gets the current transmit auto detect instrumentation configuration of the ATM port with ID *portID* on card *cardID*, chassis *chasID*. This command uses the queue ID to specify the correct queue. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

Call this command before calling **autoDetectInstrumentation cget option** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

autoDetectInstrumentation **getRx *chasID cardID portID***

Gets the current receive auto detect instrumentation configuration of the port with ID *portID* on card *cardID*, chassis *chasID*. Call this command before calling **autoDetectInstrumentation cget option** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

autoDetectInstrumentation **getTx *chasID cardID portID [streamID sequenceType]***

Gets the current transmit auto detect instrumentation configuration of the port with ID *portID* on card *cardID*, chassis *chasID*. This command can also use the stream ID. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

Call this command before calling **autoDetectInstrumentation cget option** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

autoDetectInstrumentation **setCircuitTx *chasID cardID portID [circuitID] streamID***

Sets the configuration of the stream with id *streamID* on its circuit *circuitID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL by reading the configuration option values set by the **autoDetectInstrumentation config option value** command.

autoDetectInstrumentation **setDefaults**

Sets to IxTclHal default values for all configuration options.

autoDetectInstrumentation `setQueueTx` *chasID cardID portID [queueID] [streamID]* [*sequenceType*]

Sets the transmit auto detect instrumentation configuration on the ATM port with ID *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **autoDetectInstrumentation** config *option* command. This command uses queue ID to specify which ATM queue on the port should be used. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

autoDetectInstrumentation `setRx` *chasID cardID portID*

Sets the receive auto detect instrumentation configuration of the port with ID *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **autoDetectInstrumentation** config *option* command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

autoDetectInstrumentation `setTx` *chasID cardID portID [streamID sequenceType]*

Sets the transmit auto detect instrumentation configuration of the port with ID *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **autoDetectInstrumentation** config *option* command. This command can also use the stream ID. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

EXAMPLES

package req IxTclHal

```
set hostname woodstock

if { [ixConnectToChassis $hostname] } {
    errorMsg "error connecting $hostname chassis"
    return "FAIL"
}

set chassisId      [chassis cget -id]
set cardId        2
set portId         1

set portList [list [list $chassisId $cardId $portId] ]

if { [port get $chassisId $cardId $portId] } {
    errorMsg "error getting port on $chassisId $cardId $portId"
}

if { [port isValidFeature $chassisId $cardId $portId
$::portFeatureAutoDetectRx] } {
    port config -enableAutoDetectInstrumentation $::true
    if {[port set $chassisId $cardId $portId]} {
        errorMsg "error setting port on \
$chassisId $cardId $portId"
    }
    autoDetectInstrumentation setDefault
    autoDetectInstrumentation config -startOfScan 26
    autoDetectInstrumentation config -signature \
"33 44 44 44 44 44 44 44 44 44 44 44 44 44 66"
    autoDetectInstrumentation config \
-enableSignatureMask $::true
    autoDetectInstrumentation config -signatureMask \
"AA AA AA AA AA AA AA AA AA AA"
    autoDetectInstrumentation setRx $chassisId \
$cardId $portId
    ixWritePortsToHardware portList
} else {
    errorMsg " portFeatureAutoDetectRx is not supported on \
port $chassisId $cardId $portId "
```

SEE ALSO

NAME - basicLinkServices

basicLinkServices-configure Basic Link Service protocols over fibre channel

SYNOPSIS

basicLinkServices *sub-command options*

DESCRIPTION

Basic Link Services are single frame, single sequence commands that are embedded in an unrelated exchange. Basic Link Services commands consist of only a single Basic Link_Data frame and are interspersed or are a part of a Sequence for an Exchange performing a specific protocol other than Basic Link Service. Basic Link Service commands support low-level functions and login is not required prior to using such commands.

STANDARD OPTIONS

bisCommandCode

The Command Code list contains the Basic Link Service commands.

The options are:

Option	Usage
<i>basicAccept</i>	Basic Accept is a single frame Link Service Reply Sequence that notifies the transmitter of a Basic Link Service Request frame that the request has been completed.
<i>basicReject</i>	Basic Reject is a single frame Link Service Reply Sequence that notifies the transmitter of a Basic Link Service Request frame that the request has been rejected.
<i>abortSequence</i>	Abort Sequence (ABTS) frame is used by the Sequence Initiator to request that the Sequence Recipient abort one or more Sequences and by the Sequence Recipient to request that the ABTS Recipient abort the entire Exchange.

basicAccept

The basicAccept options are:

Option	Usage
<i>transferSequenceInitiative</i>	If true, the Basic Accept Link Service Reply Sequence transfers the Sequence Initiative by setting the Sequence Initiative bit (Bit 16) to one in F_CTL on the last Data frame of the Reply Sequence.
<i>abortingEntireExchange</i>	If true, aborts the transfer of Sequence Initiative.
<i>hasInformationOnLastDeliverableSequence</i>	If true, provides information about the last delivered Sequence Initiative.

Option	Usage
<i>originatorExchangeId</i>	The Originator assigns each new Exchange an Originator Exchange ID (OX_ID) unique to the Originator or Originator-Responder pair and embeds it in all frames of the Exchange.
<i>responderExchangeId</i>	ResponderExchangeIdExchange Identifiers are used to uniquely identify an Exchange. The Responder assigns Responder ID (RX_ID) that is unique to the Responder or Responder-Originator pair and communicates it to the Originator before the end of the first Sequence of the Exchange.
<i>lastDeliverableSequenceId</i>	Sets the last deliverable Sequence Identifier assigned by the Sequence Initiator.
<i>sequenceIdValidity</i>	The value validating the Sequence Identifier.
<i>lowSequenceCount</i>	Indicates low Sequence Count. The sequence count (SEQ_CNT) is a two-byte field that indicates the sequential order of Data frame transmission within a single Sequence or multiple consecutive Sequences for the same Exchange.
<i>highSequenceCount</i>	Indicates high Sequence Count.

basicReject

The basicReject options are:

Option	Usage
<i>reasonCode</i>	The Basic Reject reason codes are <i>invalidCommandCode</i> , <i>logicalError</i> , <i>logicalBusy</i> , <i>protocolError</i> , and <i>unableToPerformCommandRequest</i> .
<i>reasonCodeExplanation</i>	The Basic Reject reason codes explanation are <i>noAdditionalExplanation</i> , <i>invalidOxId-RxIdCombination</i> , and <i>sequenceAborted</i> .
<i>vendorSpecificCode</i>	Specification of the referenced item is determined by the SCSI device vendor. The default value is 0.

abortSequence

Abort Sequence (ABTS) frame is used by the Sequence Initiator to request that the Sequence Recipient abort one or more Sequences and by the Sequence Recipient to request that the ABTS Recipient abort the entire Exchange.

EXAMPLES

See under [fibreChannel](#).

SEE ALSO

[fibreChannel](#)

NAME - bert

bert - configure Packet over Sonet cards for Bit Error Rate Testing.

SYNOPSIS

bert *sub-command options*

DESCRIPTION

The **bert** command is used to configure the transmission and receive patterns for BERT testing. Deliberate errors may be inserted with the *bertErrorGeneration* command. Refer to the *Ixia Reference Guide* for a discussion on BERT testing in Ixia equipment.

bert commands operate on concatenated and channelized cards. Cards capable of channelization must be put in that mode by setting the *port* command's *transmitMode* setting to *portTxModeBertChannelized*. They can be further channelized by using the **bert channelize** sub-command. Channel selection is accomplished with the optional *level* argument in the *set* and *get* commands. See *bert* and *bertErrorGeneration* on page 5-27 for more details on level selection.

STANDARD OPTIONS

enableInvertRxPattern
enable / disable

If *txRxPatternMode* is set to *independent*, this indicates that the expected receive pattern is to be inverted. (*default = disable*)

enableInvertTxPattern
enable / disable

If set, indicates that the transmitted pattern is to be inverted. (*default = disable*)

enableStats
enable / disable

Only applicable when *portFeatureBertList* is active. If set, enables BERT lane statistics to be collected. (*default = disable*)

rxPatternIndex

If *txRxPatternMode* is set to *independent*, this indicates the expected receive pattern: one of a set of predefined patterns:

Option	Value	Usage
<i>bertPatternAllZero</i>	8	all zeroes are expected.
<i>bertPatternAlternatingOneZero</i>	9	alternating ones and zeroes are expected.
<i>bertPatternUserDefined</i>	10	the pattern indicated in <i>rxUserPattern</i> is expected, but inverted
<i>bertPattern2_11</i>	12	the 2^{11} pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_15</i>	13	the 2^{15} pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_20</i>	14	the 2^{20} pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_23</i>	15	the 2^{23} pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_31</i>	11	the 2^{23} pattern as specified in ITU-T 0151 is expected.
<i>bertPatternAutoDetect</i>	32	(<i>default</i>) the pattern is automatically detected by the receiver.

Option	Value	Usage
<i>bertPattern2_7</i>	24	the 2^7 pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_9</i>	24	the 2^9 pattern as specified in ITU-T 0151 is expected.
<i>bertPatternLowFreq</i>	26	a low frequency pattern
<i>bertPatternHighFreq</i>	27	a high frequency pattern
<i>bertPatternContinuousRandom</i>	28	a continuous random pattern
<i>bertPatternContinuousJitter</i>	29	a continuous jitter pattern
<i>bertPatternLaneDetect</i>	31	used to detect the lane pattern and how the lanes are connected between ports

rxUserPattern

If the *rxPatternIndex* is set to *user defined*, then this is the expected pattern. If the pattern is shorter than the received data, then the pattern is repeated as necessary. If the pattern is not suitable for use (especially in unframed BERT), then a message is logged to the error file, along with a correct value. (*default = 00 00 00 00*)

txPatternIndex

Indicates the pattern to be transmitted: one of a set of predefined patterns:

Option	Value	Usage
<i>bertPatternAllZero</i>	8	all zeroes are expected.
<i>bertPatternAlternatingOneZero</i>	9	alternating ones and zeroes are expected.
<i>bertPatternUserDefined</i>	10	the pattern indicated in <i>rxUserPattern</i> is expected, but inverted
<i>bertPattern2_11</i>	12	the 2^{11} pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_15</i>	13	the 2^{15} pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_20</i>	14	the 2^{20} pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_23</i>	15	the 2^{23} pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_31</i>	11	the 2^{23} pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_7</i>	24	the 2^7 pattern as specified in ITU-T 0151 is expected.
<i>bertPattern2_9</i>	24	the 2^9 pattern as specified in ITU-T 0151 is expected.
<i>bertPatternLowFreq</i>	26	a low frequency pattern
<i>bertPatternHighFreq</i>	27	a high frequency pattern
<i>bertPatternContinuousRandom</i>	28	a continuous random pattern
<i>bertPatternContinuousJitter</i>	29	a continuous jitter pattern
<i>bertPatternLaneDetect</i>		

txRxPatternMode

Indicates if transmit and receive patterns are tied together or not:

Option	Value	Usage
<i>bertTxRxCoupled</i>	0	the <i>rxPatternIndex</i> , <i>rxUserPattern</i> and <i>enabInvertRxPattern</i> values are set from <i>txPatternIndex</i> , <i>txUserPattern</i> and <i>enabInvertTxPattern</i> .
<i>bertTxRxIndependent</i>	1	(default) transmit and receive patterns are set independently.

txUserPattern

If the *txPatternIndex* is set to *user defined*, then this is the transmitted pattern. If the pattern is shorter than the packet data size, then the pattern is repeated as necessary. (default = 00 00 00 00)

COMMANDS

The **bert** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

bert cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **bert** command.

bert channelize *chasID cardID portID [level]*

Causes the level indicated by the *level* parameter of the indicated port to be channelized. The first level of channelization occurs when the card is set in channelized mode using the *port* command's *transmitMode* variable to *portTxModeBertChannelized*. Second and subsequent levels may be channelized with this command. For example, in an OC192 Channelized BERT card, the first OC48 channel is known as *1.0*. It may be channelized by using:

```
bert channelize 1 2 1 1.0
```

The *level* parameter is expressed as a floating point number for all load modules except the 10GE XAUI module, where it must always be an integer (for example, 1, 2, 3 or 4).

bert config *option value*

Modify the configuration options of the bert. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for bert.

bert get *chasID cardID portID laneNo [level]*

Gets the current configuration of the bert for port with id *portID* on card *cardID*, chassis *chasID* from its hardware. If the card is channelized, then the optional *level* parameter must be used to select the appropriate channel. If the card is 40GE LSM XMV or 100GE LSM XMV, the *laneNumber* option is used to specify the BERT lane. Call this command before calling **bert cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

bert isChannelized *chassisID cardID portID [level]*

Returns 1 if the requested channel for the indicated port is channelized and 0 otherwise.

bert set *chassisID cardID portID laneNo [level]*

Sets the configuration of the bert in IxHAL for port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **bert config *option value*** command. If the card is channelized, then the optional *level* parameter must be used to select the appropriate channel. The *level* parameter is expressed as a floating point number for all load modules except the 10GE XAUI module, where it must always be an integer (for example, 1, 2, 3 or 4). If the card is 40GE LSM XMV or 100GE LSM XMV, the *laneNumber* option is used to specify the BERT lane.

Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Bert is not supported for this port type (PoS only)
- Configured parameters are not valid for this setting

bert setDefault

Sets to IxTclHal default values for all configuration options.

bert unChannelize *chassisID cardID portID level*

Causes the level indicated by the *level* parameter of the indicated port to be unchannelized. The *level* parameter is expressed as a floating point number for all load modules except the 10GE XAUI module, where it must always be an integer (for example, 1, 2, 3 or 4).

EXAMPLES

```
package require IxTclHal

#####
#
# First section works with an OC48c Bert card in slot 22
#
#####

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```

}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that an OC48c BERT card is in slot 22
set card 22
set portList [list [list $chassis $card 1]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Check for missing card
if {[card get $chassis $card] != 0} \
{
    ixPuts "Card $card does not exist"
    return 1
}

# Get the type of card and check if it's the correct type
set cardType [card cget -type]
if {[port isValidFeature $chassis $card 1 portFeatureBert] == 0} \
{
    ixPuts "Card $card does not have Bert capability"
    return 1
}

# Set the options to default values
bert setDefault

# In this example, we'll couple the transmit and receive side
# To simulate the port connected to a device which sends its data
# back to the port
bert config -txRxPatternMode bertTxRxCoupled

# Select inverted 2^20 pattern to transmit
bert config -txPatternIndex bertPattern2_20
bert config -enableInvertTxPattern enable
bert set $chassis $card 1
ixWritePortsToHardware portList

# Now we need to send a start transmit to the port to gather
# statistics
# and then read the statistics
ixStartPortTransmit $chassis $card 1
after 1000
# Stop statistics gathering
ixStopPortTransmit $chassis $card 1

# Fetch the number of bits received
stat get statBertBitsReceived $chassis $card 1
set received [stat cget -bertBitsReceived]
ixPuts "$received bits were received after 1 second"

# Bert error generation example

```

```

bertErrorGeneration setDefault

# Set for 10^4 errors
bertErrorGeneration config -errorBitRate bert_1e4
bertErrorGeneration set $chassis $card 1
ixWritePortsToHardware portList

# Enable statistics gathering
ixStartPortTransmit $chassis $card 1

# Send the error continuously for 10 seconds
bertErrorGeneration startContinuousError $chassis $card 1
after 10000
bertErrorGeneration stopContinuousError $chassis $card 1
ixStopPortTransmit $chassis $card 1

# And get the number of errored bits
stat get statBertBitErrorsReceived $chassis $card 1
set received [stat cget -bertBitErrorsReceived]
ixPuts "$received bit errors were received after 10 seconds"

#####
#
# Second section works with an OC192/10GE/BERT card in slot 51
# In order to demonstrate channelized BERT operation
#
#####

set card 51
set portList [list [list $chassis $card 1]]

# Check for missing card
if {[card get $chassis $card] != 0} \
{
    ixPuts "Card $card does not exist"
    return 1
}

# Get the type of card and check if it's the correct type
set cardType [card cget -type]
if {[port isValidFeature $chassis $card 1 portFeatureBert] == 0} \
{
    ixPuts "Card $card does not have Bert capability"
    return 1
}

# Set port to channelized
port setFactoryDefaults $chassis $card 1
port config -transmitMode portTxModeBertChannelized
port config -receiveMode portRxModeBertChannelized
if [port set $chassis $card 1] {
    ixPuts "Could not port set on $chassis:$card:1"
    return 1
}

# Set the options channelize the second OC48 channel
bert setDefault
if [bert channelize $chassis $card 1 2.0] {
    ixPuts "Could not channelize $chassis:$card:1 2.0"
    return 1
}

# couple the transmit and receive side

```

```

bert config -txRxPatternMode bertTxRxCoupled

# Select alternating one, zero pattern
bert config -txPatternIndex bertPatternAlternatingOneZero

# Set the characteristics for the third OC12 channel on the second
OC48 channel
if [bert set $chas $card 1 2.3] {
    ixPuts "bert set failed on $chas:$card:1 level 2.3"
    return 1
}

# Use isChannelized to make sure this worked
if {[bert isChannelized $chas $card 1 2.0] == 0} {
    ixPuts "Channel 2.0 is not channelized"
}
ixWritePortsToHardware portList

# Now we need to send a start transmit to the port to gather
statistics
# and then read the statistics
ixStartPortTransmit $chas $card 1
after 1000
# Stop statistics gathering
ixStopPortTransmit $chas $card 1

# Fetch the number of bits received on the specific channel
stat getBertChannel $chas $card 1 2.3
stat get statBertBitsReceived $chas $card 1
set received [stat cget -bertBitsReceived]
ixPuts "$received bits were received after 1 second"

# Bert error generation example
bertErrorGeneration setDefault

# Set for 10^4 errors
bertErrorGeneration config -errorBitRate bert_1e4
bertErrorGeneration set $chas $card 1
ixWritePortsToHardware portList

# Enable statistics gathering
ixStartPortTransmit $chas $card 1

ixPuts "Starting error generation"
# Send the error continuously for 10 seconds
bertErrorGeneration startContinuousError $chas $card 1 2.3
after 10000
bertErrorGeneration stopContinuousError $chas $card 1 2.3
ixStopPortTransmit $chas $card 1

# And get the number of errored bits
stat get statBertBitErrorsReceived $chas $card 1
set received [stat cget -bertBitErrorsReceived]
ixPuts "$received bit errors were received after 10 seconds"

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

bertErrorGeneration

NAME - bertErrorGeneration

bertErrorGeneration - configure the BERT Error Generation parameters on a Packet over Sonet port of a card on a chassis.

SYNOPSIS	bertErrorGeneration <i>sub-command options</i>
DESCRIPTION	The bertErrorGeneration command is used to configure the insertion of deliberate errors on a port. The port must previously have been setup using the <i>bert</i> command. Refer to the <i>Ixia Reference Guide</i> for a discussion on BERT testing in Ixia equipment. bertErrorGeneration commands operate on concatenated and channelized cards. Cards capable of channelization must be put in that mode by setting the <i>port</i> command's <i>transmitMode</i> setting to <i>portTxModeBertChannelized</i> . They can be further channelized by using the bert channelize sub-command. Channel selection is accomplished with the optional <i>level</i> argument in the <i>set</i> and <i>get</i> commands. See <i>bert and bertErrorGeneration</i> on page 5-27 for more details on level selection.
STANDARD OPTIONS	
bitMask	For OC-48 unframed BERT: a 32-bit mask, expressed as a list of four one-byte elements, which indicates which bit in a 32-bit word is to be errored. (<i>default</i> = <i>00000000 00000000 00000000 00000001</i>) For all other BERT: a 128-bit mask, expressed as a list of 16 two-byte hex elements, which indicates which bit in a 128-bit word is to be errored. (<i>default</i> = <i>00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00</i>)
burstCount	The number of times that the error is to be inserted. (<i>default</i> = <i>1</i>)
burstPeriod	The number of bits between error insertions. (<i>default</i> = <i>128</i>)
burstWidth	The number of bits in the error insertion; this should be set to 32 or less. (<i>default</i> = <i>128</i>)
continuousErrorInsert true/false	Inserts BERT errors continuously, at the rate selected in the <i>errorBitRate</i> option. (<i>default</i> = <i>false</i>)
errorBitRate	During continuous burst rate situations, this is the error rate.

Option	Value	Usage
bert_1e2	0	An error is inserted every 2^2 (4) bits.
bert_1e3	1	An error is inserted every 2^3 (8) bits.
bert_1e4	2	An error is inserted every 2^4 (16) bits.
bert_1e5	3	An error is inserted every 2^5 (32) bits.
bert_1e6	4	An error is inserted every 2^6 (64) bits.
bert_1e7	5	An error is inserted every 2^7 (128) bits.
bert_1e8	6	An error is inserted every 2^8 (256) bits.

Option	Value	Usage
bert_1e9	7	(default) An error is inserted every 2^9 (512) bits.
bert_1e10	8	An error is inserted every 2^{10} (1024) bits.
bert_1e11	9	An error is inserted every 2^{11} (2048) bits.
bert_UserDefined	10	An error is inserted every <i>period</i> bits.

period

If *errorBitRate* is set to *bert_UserDefined*, then this is the number of bits between error insertions. (*default* = 4000000000)

COMMANDS

The **bertErrorGeneration** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

bertErrorGeneration cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **bertErrorGeneration** command.

bertErrorGeneration config option value

Modify the configuration options of the bertErrorGeneration. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for bertErrorGeneration.

bertErrorGeneration get chasID cardID portID laneNo [level]

Gets the current configuration of the bertErrorGeneration for port with id *portID* on card *cardID*, chassis *chasID* from its hardware. If the card is channelized, then the optional *level* parameter must be used to select the appropriate channel. The *laneNo* option is only applicable when *portFeatureBertList* is active.

Call this command before calling **bertErrorGeneration cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

bertErrorGeneration insertSingleError chasID cardID portID laneNumber [level]

Inserts a single error into the BERT stream as specified by the STANDARD OPTIONS. If the card is channelized, then the optional *level* parameter must be used to select the appropriate channel. The *laneNumber* option is only applicable when *portFeatureBertList* is active, and is used to specify the BERT lane where the error is generated. Specific errors are:

- No connection to a chassis
- Invalid port number

bertErrorGeneration set chasID cardID portID laneNo [level]

Sets the configuration of the bertErrorGeneration in IxHAL for port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **bertErrorGeneration config option value** command. If the card is channelized, then the optional *level* parameter must be used to select the

appropriate channel. The *laneNo* option is only applicable when *portFeatureBertList* is active.

Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Bert is not supported for this port type (PoS only)
- Configured parameters are not valid for this setting

bertErrorGeneration `setDefault`

Sets to IxTclHal default values for all configuration options.

bertErrorGeneration `startContinuousError` *chasID cardID portID laneNumber [level]*

Continuously inserts errors into the BERT stream, as dictated by the STANDARD OPTIONS. If the card is channelized, then the optional *level* parameter must be used to select the appropriate channel. The *laneNumber* option is only applicable when *portFeatureBertList* is active, and is used to specify the BERT lane where the error is generated. Errors are inserted once every $2^{errorBitRate}$ bits. Specific errors are:

- No connection to a chassis
- Invalid port number
- Bert is not supported for this port type (PoS only)

bertErrorGeneration `stopContinuousError` *chasID cardID portID laneNumber [level]*

Stops the continuous insertion of errors into the BERT stream. If the card is channelized, then the optional *level* parameter must be used to select the appropriate channel. The *laneNumber* option is only applicable when *portFeatureBertList* is active, and is used to specify the BERT lane where the error is generated.

Specific errors are:

- No connection to a chassis
- Invalid port number
- Bert is not supported for this port type (PoS only)

EXAMPLES

See examples in [bert](#).

SEE ALSO

[bert](#)

NAME - bertUnframed

bertUnframed - configure unframed BERT specific parameters on a port

SYNOPSIS	bertUnframed <i>sub-command options</i>	
DESCRIPTION	The bertUnframed command is used to configure line speed and other operational characteristics of an unframed BERT port.	
STANDARD OPTIONS		
dataRate	The particular data rate for the port. The choices available depend on the setting of the <i>clockSelect</i> option of the <i>card</i> command. Options include:	
Option	Value	Usage
bertUnframedInvalidLineRate	0	(default)
bertUnframedOc3	1	155.52 Mbps (OC-3). Only valid if <i>card - cardSelect = cardBertUnframedClockSonet</i> .
bertUnframedOc12	2	622.08 Mbps (OC-12). Only valid if <i>card - cardSelect = cardBertUnframedClockSonet</i> .
bertUnframedOc48	3	2.488 Gbps (OC-48). Only valid if <i>card - cardSelect = cardBertUnframedClockSonet</i> .
bertUnframedOc3WithFec	4	166.63 Mbps (OC-3 FEC). Only valid if <i>card - cardSelect = cardBertUnframedClockSonetWithFEC</i>
bertUnframedOc12WithFec	5	666.51 Mbps (OC-12 FEC). Only valid if <i>card - cardSelect = cardBertUnframedClockSonetWithFEC</i>
bertUnframedOc48WithFec	6	2.67 Gbps (OC-48 FEC). Only valid if <i>card - cardSelect = cardBertUnframedClockSonetWithFEC</i>
bertUnframedGigEthernet	7	1.25Gbps (Gigabit Ethernet). Only valid if <i>card - cardSelect = cardBertUnframedClockGigE</i>
bertUnframedFiberChannel1	8	1.062 Gbps (Fibre Channel). Only valid if <i>card - cardSelect = cardBertUnframedClockFiberChannel</i>
bertUnframedFiberChannel2	9	2.124 Gbps (2x Fibre Channel). Only valid if <i>card - cardSelect = cardBertUnframedClockFiberChannel</i>
bertUnframed1x	10	Use the external clock directly. Only valid if <i>card - cardSelect = cardBertUnframedClockExternal</i>
bertUnframed4x	11	Multiply the external clock rate by a factor of 4. Only valid if <i>card - cardSelect = cardBertUnframedClockExternal</i>
bertUnframed8x	12	Multiply the external clock rate by a factor of 8. Only valid if <i>card - cardSelect = cardBertUnframedClockExternal</i>

Option	Value	Usage
bertUnframed16x	13	Multiply the external clock rate by a factor of 16. Only valid if <i>card - cardSelect = cardBertUnframedClockExternal</i>

enableTransceiver
BypassPLLExClock

(*default = 0*)

operation

The basic line operation.

Option	Value	Usage
bertUnframedNormal	0	(<i>default</i>) Operate in normal transmit, receive mode.
bertUnframedDiagnostic Loopback	1	Operate in diagnostic loopback mode.
bertUnframedLineLoopback	2	Operate in line loopback mode.

useRecoveredClock
true / false

If *true*, use the clock recovered from the received data rather than the internally generated clock. (*default = 0*)

COMMANDS

The **bertUnframed** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

bertUnframed cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **bertUnframed** command.

bertUnframed config *option value*

Modify the configuration options of the **bertUnframed** command. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **bertUnframed**.

bertUnframed get *chasID cardID portID*

Gets the current configuration for port with id *portID* on card *cardID*, chassis *chasID*, from its hardware. Call this command before calling **bertUnframed cget** *option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

bertUnframed set *chasID cardID portID*

Sets the configuration of the bertUnframed command in IxHAL for port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **bertUnframed config** *option value* command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Bert is not supported for this port type (PoS only)
- Configured parameters are not valid for this setting

bertUnframed setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that a multi-rate unframed Bert card in slot 25
set card 25
set portList [list [list $chas $card 1]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Check for missing card
if {[card get $chas $card] != 0} \
{
    ixPuts "Card $card does not exist"
    return 1
}

# Get the type of card and check if it's the correct type
set cardType [card cget -type]
if {[port isValidFeature $chas $card 1 portFeatureBertUnframed] == 0} \
{
    ixPuts "Card $card does not have unframed Bert capability"
    return 1
}

# Set the card clock select to Sonet FEC
card setDefault
card config -clockSelect cardBertUnframedClockSonetWithFEC
```

```

if [card set $chas $card] {
    ixPuts "Can't card set for $chas:$card"
    return 1
}

# Set the options to default values
bertUnframed setDefault

# Set the unframed speed to OC48c FEC rates and normal operation
bertUnframed config -dataRate bertUnframedOc48WithFec
bertUnframed config -operation bertUnframedNormal

if [bertUnframed set $chas $card 1] {
    ixPuts "Could not bertUnframed set on $chas:$card:1"
    return 1
}

# Now do the normal Bert testing things
bert setDefault
bert config -txRxPatternMode bertTxRxCoupled
bert config -txPatternIndex bertPattern2_20
bert config -enableInvertTxPattern enable

if [bert set $chas $card 1] {
    ixPuts "Can't bert set on $chas:$card:1"
    return 1
}
ixWritePortsToHardware portList

# Now we need to send a start transmit to the port to gather
statistics
# and then read the statistics
ixStartPortTransmit $chas $card 1
after 1000
# Stop statistics gathering
ixStopPortTransmit $chas $card 1

# Fetch the number of bits received
stat get statBertBitsReceived $chas $card 1
set received [stat cget -bertBitsReceived]
ixPuts "$received bits were received after 1 second"

# Bert error generation example
bertErrorGeneration setDefault

# Set for 10^4 errors
bertErrorGeneration config -errorBitRate bert_1e4
bertErrorGeneration set $chas $card 1
ixWritePortsToHardware portList

# Enable statistics gathering
ixStartPortTransmit $chas $card 1

# Send the error continuously for 10 seconds
bertErrorGeneration startContinuousError $chas $card 1
after 10000
bertErrorGeneration stopContinuousError $chas $card 1
ixStopPortTransmit $chas $card 1

# And get the number of errored bits
stat get statBertBitErrorsReceived $chas $card 1
set received [stat cget -bertBitErrorsReceived]
ixPuts "$received bit errors were received after 10 seconds"

```

```
# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

bert, bertErrorGeneration, card, port

NAME - capture

capture - configure the capture parameters on a port of a card on a chassis.

SYNOPSIS

capture *sub-command options*

DESCRIPTION

The **capture** command is used to configure the capture parameters and sets up the capture buffer. The *afterTriggerFilter*, *beforeTriggerFilter*, *captureMode*, *continuousFilter*, *fullAction* and *triggerPosition* options are associated with the circular buffer feature which is only available on some card types. Refer to the *Ixia Hardware Guide* for a list of which modules support the features.

The capture process itself is started through the use of the *portGroup setCommand startCapture* command, or the *ixStartCapture* high-level command. The capture is stopped with the use of the *portGroup setCommand stopCapture* command, or the *ixStopCapture* high-level command, or a *captureBuffer get* command. That is, the act of reading the capture buffer stops the capture process. The high-level command, *ixCheckTransmitDone*, may be used to wait until all ports have finished transmitting.

STANDARD OPTIONS

afterTriggerFilter

Controls the capture of data after triggering when operating in triggered mode (*captureMode = captureTriggerMode*). Available option values are:

Option	Value	Usage
<i>captureAfterTriggerAll</i>	0	capture all data after trigger.
<i>captureAfterTrigger Filter</i>	1	(default) capture filtered data after trigger using filter settings.
<i>captureAfterTrigger ConditionFilter</i>	2	capture filtered data after trigger as per the filter settings.

beforeTriggerFilter

Controls the capture of data prior to triggering when operating in triggered mode (*captureMode = captureTriggerMode*). Available option values are:

Option	Value	Usage
<i>captureBeforeTriggerAll</i>	0	capture all data before trigger.
<i>captureBeforeTrigger None</i>	1	(default) capture none of the data before trigger.
<i>captureBeforeTrigger Filter</i>	2	capture filtered data before trigger as per the filter settings.

captureMode

Controls whether data capture is performed in a continuous or triggered mode. Available option values are:

Option	Value	Usage
<i>captureContinuous Mode</i>	0	capture data in the buffer continuously, regardless of trigger settings. Data may be filtered; see <i>continuousFilter</i> . When the buffer is full, begin storing new frames at the end of the buffer over-writing the previously stored frames.

Option	Value	Usage
<i>captureTriggerMode</i>	1	(default) capture data only after triggered. After the buffer is full, do not capture any more frames.

continuousFilter

Controls whether data captured in continuous mode (*captureMode = captureContinuousMode*) is filtered or not. Available option values are:

Option	Value	Usage
<i>captureContinuousAll</i>	0	(default) capture all data, regardless of filter settings.
<i>captureContinuousFilter</i>	1	capture only filtered data, as per filter settings.

enableSmallPacketCapture true/false

Applies to OC12 cards only. Capture of packets of 48 bytes or less at full wire rates can be problematic and is usually treated as an error. This setting allows packets of 48 bytes or less in length to be captured. The data captured, however, may be corrupt. (default = false)

fullAction

Used for LM100Tx boards only. Controls the action of the buffer when it reaches the full status. Available option values are:

Option	Value	Usage
<i>lock</i>	0	(default) after the buffer is full, do not capture any more frames
<i>wrap</i>	1	when the buffer is full, start storing the new frames at the beginning of the buffer over-writing the previously stored frames

nPackets

Read-only. Number of packets available or captured in the capture buffer.

sliceSize

The maximum number of octets of each frame that is saved in this capture buffer. For example, if a 1500 octet frame is received by the probe and this option is set to 500, then only 500 octets of the frame is stored in the associated capture buffer. If this option is set to 0, the capture buffer saves as many octets as is possible. If the *sliceSize* is set larger than the maximum hardware supported slice size, the maximum is used. (default = 8191)

triggerPosition

Controls the dividing line within the capture buffer between before trigger data and post trigger data. This control is only useful in triggered mode (*captureMode = captureTriggerMode*) and before trigger capture enabled (*beforeTriggerFilter = captureBeforeTriggerAll* or *captureBeforeTriggerFilter*). *TriggerPosition* is expressed as a percentage of the total buffer size. The beginning of the buffer with this percentage is used in a wrap-around mode for before trigger data and the remainder is filled up with triggered data. (default = 1.0)

DEPRECATED STANDARD OPTIONS
filter
sliceOffset
trigger

COMMANDS

The **capture** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

capture cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **capture** command.

capture config option value

Modify the configuration options of the capture. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for capture.

capture get chasID cardID portID

Gets the current configuration of the capture for port with id *portID* on card *cardID*, chassis *chasID*. from its hardware. Call this command before calling **capture cget option value** to get the value of the configuration option. In order for this command to succeed, the port must either be unowned, or you must be logged in as the owner of the port. Specific errors are:

- No connection to a chassis
- Invalid port number

capture set chasID cardID portID

Sets the configuration of the capture in IxHAL for port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **capture config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

capture setDefault

Sets to IxTclHal default values for all configuration options.

capture write chasID cardID portID

Writes or commits the changes in IxHAL to hardware for the capture related parameters on port with id *portID* on card *cardID*, chassis *chasID*. Before using this command, use the *capture set*. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Network problem between the client and chassis

EXAMPLES

```
package require IxTclHal

set host techpubs-400
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
```

```

        if [ixConnectToTclServer $host] {
            ixPuts "Could not connect to $host"
            return 1
        }
    }

    # Now connect to the chassis
    if [ixConnectToChassis $host] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Get the chassis ID to use in port lists
    set chas [ixGetChassisID $host]

    # Assume that there's a four port 10/100 TXS card in this slot
    # with port 1 looped to port 2
    set card 1
    set portlist [list [list $chas $card 1] [list $chas $card 2]]
    set txPortList [list [list $chas $card 1]]
    set rxPortList [list [list $chas $card 2]]

    # Login before taking ownership
    if [ixLogin $username] {
        ixPuts $::ixErrorInfo
        return 1
    }
    # Take ownership of the ports we'll use
    if [ixTakeOwnership $portList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Reset Ports for Factory Defaults Stream Mode / Capture
    port setFactoryDefaults $chas $card 1
    port setFactoryDefaults $chas $card 2
    port setDefault
    if [port set $chas $card 1] {
        ixPuts $::ixErrorInfo
        return 1
    }
    if [port set $chas $card 2] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # The number of frames to get in the capture buffer at one time
    # It's better not to read all of the captured packets -
    # It might be very large and take a good deal of time and memory
    set frameSlice 4000

    # Put the time in the outbound stream
    stream setDefault
    stream config -sa {00 de ad be ef 00}
    stream config -da {00 ba be fa ce 00}
    stream config -dma stopStream
    stream config -numFrames 50000
    stream config -fir true
    if [stream set $chas $card 1 1] {
        ixPuts $::ixErrorInfo
    }
    if [ixWritePortsToHardware portlist] {
        ixPuts $::ixErrorInfo
    }
}

```

```

        return 1
    }

    # Wait for Link
    after 1000
    if [ixCheckLinkState portList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    if [ixClearStats rxPortList] {
        ixPuts $::ixErrorInfo
        return 1
    }
    if [ixStartCapture rxPortList] {
        ixPuts $::ixErrorInfo
        return 1
    }
    if [ixStartTransmit txPortList] {
        ixPuts $::ixErrorInfo
        return 1
    }
    if [ixCheckTransmitDone txPortList] {
        ixPuts $::ixErrorInfo
        return 1
    }
    if [ixStopCapture rxPortList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Get the number of frames captured
    if [capture get $ch $card 2] {
        ixPuts $::ixErrorInfo
        return 1
    }
    set numFrames [capture cget -npackets]
    ixPuts "$numFrames frames captured"

    # Set up jitter calculation for 64 byte packets only
    captureBuffer config -enableFramesize true
    captureBuffer config -framesize       64
    captureBuffer setConstraint {
        ixPuts "Could not set captureBuffer constraints"
    }

    # Only look at the first $frameSlice frames
    if {$numFrames > $frameSlice} {set numFrames $frameSlice}
    ixPuts "Frame\tTime\tLatency\tData"

    # Go through all of the frames $frameSlice frames at a time
    for {set frameNo 1} {$frameNo <= $numFrames} \
        {incr frameNo $frameSlice} {
        set lastFrame [expr $frameNo + $frameSlice - 1]
        if {$lastFrame > $numFrames} {$lastFrame = $numFrames}

        # Get the batch of frames
        if [captureBuffer get $ch $card 2 $frameNo $lastFrame] {
            ixPuts $::ixErrorInfo
            return 1
        }
        set numCaptured [expr $lastFrame - $frameNo +1]
    }
}

```

```
ixPuts "Average latency is [captureBuffer cget -averageLatency]"

# Go through each frame in the capture buffer starting at 1
for {set i 1} {$i < $numCaptured} {incr i} {
    # Note that the frame number starts at 1
    captureBuffer getframe $i

    # Get the actual frame data
    set data [captureBuffer cget -frame]

    # We'll only look at the first bunch of bytes
    set data [string range $data 0 50]

    # Get timestamp and latency too
    set timeStamp [captureBuffer cget -timestamp]
    set latency [captureBuffer cget -latency]
    set status 'Bad'
    ixPuts "Status is [format "%x" [captureBuffer cget -status]]"
    if {[captureBuffer cget -status] &
        $::cap10100DpmGoodPacket} {
        set status 'Good'
    }
    ixPuts -none newline [expr $frameNo + $i - 1]
    ixPuts -none newline "\t$timeStamp"
    ixPuts -none newline "\t$latency"
    ixPuts -none newline "\t$status"
    ixPuts "\t$data"
}
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
return 0
```

SEE ALSO*captureBuffer*

NAME - captureBuffer

captureBuffer - view the capture frames in the captured buffer.

SYNOPSIS

captureBuffer *sub-command options*

DESCRIPTION

After the *capture* command is used to configure the capture buffer, the *captureBuffer* command is used to get a range of frames from the capture buffer. Jitter values are calculated on those frames that meet the constraint criteria. Three different types of constraint criteria are available; ethernet type, frame size and pattern. For example, if jitter is to be calculated only for 64 byte frames, then the *framesize* option must be set to *64* and the *enableFramesize* option set to *true*. Constraints must be set with *setConstraint* prior to a *get*.

Note. For some load modules (that is, LSM10GE), it is advisable to request *captureBuffer* data in chunks. Unless both the chassis and client machines have sufficiently high available memory, they may be overloaded by captured data.

When using the *get* command to retrieve the capture buffer, the capture operation is not stopped.

STANDARD OPTIONS

averageDeviation

Read-only. 64-bit value. The average deviation of the average latencies calculated by the command *captureBuffer getStatistics*.

averageLatency

Read-only. 64-bit value. The average latency (in nanoseconds) calculated by the command *captureBuffer getStatistics*.

**enableEthernetType
*true/false***

Enables the constraint used to calculate jitter statistics. If enabled, jitter is calculated only for frames whose frame type field matches the ethernet type set by the option *ethernetType*. Does not apply unless the command *captureBuffer setConstraint* is applied. (*default = false*)

**enableFramesize
*true/false***

Enables this constraint used to calculate jitter statistics. If enabled, jitter is calculated only for frames whose size matches the framesize set by the command option *framesize*. Does not apply unless the command *captureBuffer setConstraint* is applied. (*default = false*)

**enablePattern
*true/false***

Constrain the jitter statistics calculations to frames in the capture buffer that match the ethernet pattern set by the *pattern* and *patternOffset* options. Does not apply unless *captureBuffer setConstraint* is applied. (*default = false*)

etherType

Constrain the jitter statistics calculations to frames in the capture buffer that match the ethernet type set by this option. A value such as *{08 00}* would be appropriate. Does not apply unless option *enableEthernetType* is set to *true* and *captureBuffer setConstraint* is applied. (*default = “”*)

fir

Read-only. The frame identity record.

frame	<i>Read-only.</i> The contents of the selected frame based on the sliceSize parameter set by the <i>capture</i> command.
framesize	Constrain the jitter statistics calculations to frames in the capture buffer whose frame size matches the value set by this option. Does not apply unless the option <i>enableFramesize</i> is set to <i>true</i> and <i>captureBuffer setConstraint</i> is applied. <i>(default = 64)</i>
latency	<i>Read-only. 64-bit value.</i> The frame latency, calculated as the difference between the transmit time and receive time of the frame, in nanoseconds.
length	<i>Read-only.</i> The total length of the frame, regardless of the actual number of bytes in the capture buffer.
maxLatency	<i>Read-only. 64-bit value.</i> The maximum frame latency (in nanoseconds) calculated by the command <i>captureBuffer getStatistics</i> .
minLatency	<i>Read-only. 64-bit value.</i> The minimum frame latency (in nanoseconds) calculated by the command <i>captureBuffer getStatistics</i> .
numFrames	<i>Read-only.</i> The number of frames (or slices, a slice could contain a whole frame or a part of a frame) in the capture buffer. When <i>captureBuffer setConstraint</i> is called this value is updated with the number of frames for each constraint.
pattern	Enables this constraint used to calculate jitter statistics. If enabled, jitter is calculated only for frames whose pattern matches the pattern in the frame at the offset set by the command option <i>patternOffset</i> . A value of the form {11 12 02 44} would be appropriate. Does not apply unless the command <i>captureBuffer setConstraint</i> is applied. <i>(default = "")</i>
patternOffset	Used in conjunction with the <i>pattern</i> command. Does not apply unless the command <i>captureBuffer setConstraint</i> is applied. <i>(default = 12)</i>
standardDeviation	<i>Read-only. 64-bit value.</i> The standard deviation of the average latencies calculated by the command <i>captureBuffer getStatistics</i> .
status	<i>Read-only.</i> The status of the frame. Except where noted the following status values are used:

Option	Value	Usage
<i>capNoErrors</i>	0x00	captured frame has no error
<i>capBadCrcGig</i>	0x01	captured frame has a bad or missing CRC (<i>gigabit only</i>)
<i>capSymbolErrorsGig</i>	0x02	captured frame has symbol error (<i>gigabit only</i>)
<i>capBadCrcAndSymbolGig</i>	0x03	captured frame has a bad or missing CRC and symbol error (<i>gigabit only</i>)
<i>capUndersizeGig</i>	0x04	captured frame is undersize (<i>gigabit only</i>)
<i>capBadCrcAndUndersizeGig</i>	0x05	captured frame has a bad or missing CRC and is undersize (<i>gigabit only</i>)
<i>capBadCrcAndSymbolAndUndersizeGig</i>	0x07	captured frame has a bad or missing CRC and is undersize (<i>gigabit only</i>)

Option	Value	Usage
<code>capOversizeGig</code>	0x08	captured frame is oversize with a valid CRC (<i>gigabit only</i>)
<code>capBadCrc</code>	0x41	captured frame has a bad or missing CRC
<code>capBadCrcAndSymbolError</code>	0x43	captured frame has a bad or missing CRC and symbol error
<code>capUndersize</code>	0x44	captured frame is undersize
<code>capFragment</code>	0x45	captured frame is a fragment
<code>capOversize</code>	0x48	captured frame is oversize with a valid CRC
<code>capOversizeAndBadCrc</code>	0x49	captured frame is oversize with a bad or missing CRC
<code>capDribble</code>	0x50	captured frame has a dribble error
<code>capAlignmentError</code>	0x51	captured frame has alignment error (<i>10/100 only</i>)
<code>capAlignAndSymbolError</code>	0x53	captured frame has alignment and symbol error
<code>capGoodFrame</code>	0xC0	captured frame is a valid frame with no errors (<i>default</i>)
<code>capBadCrcAndGoodFrame</code>	0xC1	captured frame has a bad or missing CRC but otherwise valid frame
<code>capErrorFrame</code>	0xFF	captured frame has a general error other than one of the specified errors in this list

The following status values are used for OC48 cards:

Option	Value	Usage
<code>capOc48Trigger</code>	0x40000000	A status bit that indicates that the captured packet includes the bit that caused the trigger.
<code>capOc48GoodPacket</code>	0x80000000	captured frame is valid with no errors (
<code>capOc48TruncatedPacket</code>	0x80000001	captured frame is a truncated packet
<code>capOc48IntegritySignatureMatch</code>	0x80000008	captured frame's integrity signature matched
<code>capOc48BadIntegrityCheck</code>	0x80000010	captured frame failed data integrity validation
<code>capOc48BadTCPUDPChecksum</code>	0x80000020	captured frame has a bad TCP or UDP checksum
<code>capOc48BadIPChecksum</code>	0x80000040	captured frame is valid with no errors
<code>capOc48BadCrc</code>	0x80000080	captured frame is valid with no errors

The following status values are used for 10/100 TX and 10/100/1000 TXS cards:

Option	Value	Usage
<code>cap10100DpmTrigger</code>	0x40000000	A status bit that indicates that the captured packet includes the bit that caused the trigger.
<code>cap10100DpmGoodPacket</code>	0x80000000	captured frame is valid with no errors
<code>cap10100DpmOversize</code>	0x80000001	capture frame is oversized.
<code>cap10100DpmUndersize</code>	0x80000002	capture frame is undersized.

Option	Value	Usage
<i>cap10100DpmIntegritySignatureMatch</i>	0x80000008	captured frame's integrity signature matched
<i>cap10100DpmBadIntegrityCheck</i>	0x80000010	captured frame failed data integrity validation
<i>cap10100DpmBadTCPUDPChecksum</i>	0x80000020	captured frame has a bad TCP or UDP checksum
<i>cap10100DpmBadIPChecksum</i>	0x80000040	captured frame is valid with no errors
<i>cap10100DpmBadCrc</i>	0x80000080	captured frame is valid with no errors

The following status values are used for ATM cards and ATM/POS cards operating in ATM mode:

Option	Value	Usage
<i>capAtmEthernetBadCrc</i>	0x80000080	Bad Ethernet CRC.
<i>capAtmBadIPChecksum</i>	0x80000040	Bad IP checksum.
<i>capAtmBadTCPUDPChecksum</i>	0x80000020	Bad TCP or UDP checksum.
<i>capAtmBadIntegrityCheck</i>	0x80000010	Bad Data Integrity.
<i>capAtmIntegritySignatureMatch</i>	0x80000008	Data integrity signature matched.
<i>capAtmAal5BadCrc</i>	0x80000004	Bad AAL5 CRC.
<i>capAtmTimeout</i>	0x80000002	ATM timeout.
<i>capAtmOversize</i>	0x80000001	ATM oversize packet.
<i>capAtmGoodPacket</i>	0x80000000	Good packet received.
<i>capAtmTrigger</i>	0x40000000	Data capture was triggered.

timestamp

Read-only. 64-bit value. The arrival time of the captured frame in nanoseconds.

COMMANDS

The **captureBuffer** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

captureBuffer cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **captureBuffer** command.

captureBuffer clearConstraint

Clears the constraints used to calculate the average, standard deviation and average deviation of the latencies of the captured frames in the capture buffer. Statistics is calculated on the entire buffer.

captureBuffer config option value

Modify the configuration options of the **captureBuffer**. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **captureBuffer**.

captureBuffer export fileName [encodeVersion]

Exports the current contents of the capture buffer from the last *captureBuffer get* command to the file indicated in *fileName*; *fileName* may include a full or

relative path. The format of the file is dictated by the extension on the file (only the *.cap* and *.enc* file format is supported):

Table 2: File Formats (Export)

.txt	a text file suitable for import into a database.
.cap	a binary format for use with the <i>captureBuffer</i> import function or IxExplorer's File Import function.
.enc	<p>a binary format for use with NAI's Sniffer program. Note that when working with POS ports, the export function maps the POS frames to look like Ethernet data: the POS header is stripped off (4 bytes), the MAC address is padded out to 12 bytes with zeroes, a packet type identifier of 0x0800 (2 bytes, Ethernet) is added and the beginning of the MAC DA is overwritten with the POS header.</p> <p>Note: When a file is exported to <i>.enc</i> format, the CRC frame check sequence gets stripped away and is not present when the saved file is imported.</p>

The optional second argument is used when the *fileName*'s extension is *.enc*. The choices are:

Option	Value	Usage
<i>capExportSniffer4x</i>	2	(default) Sniffer 4.x format
<i>capExportSniffer1x</i>	3	Sniffer 1.x format

captureBuffer get *chasID cardID portID fromFrame toFrame*

Gets the group of captured frames from the capture buffer for *chasID cardID portID*, beginning with frame *fromFrame* through frame and puts it into local memory. Call this command before calling **captureBuffer getframe** *frameNum* to get the capture buffer from hardware. The **capture cget -nPackets** should be called before this command to determine how many frames are available in the capture buffer. In order for this command to succeed, the port must either be unowned, or you must be logged in as the owner of the port.

captureBuffer getConstraint *constraintNum*

Gets the constraints used to calculate the average, standard deviation and average deviation of the latencies of the captured frames in the capture buffer retrieved by **captureBuffer get**. The value returned is the constraint number. This constraint number can be used in “captureBuffer getConstraint” command to retrieve the constraint settings.

captureBuffer getframe *frameNum*

Gets the capture buffer data from local memory for *frameNum*. Call **captureBuffer get *chasID cardID portID fromFrame toFrame*** before calling this command.

captureBuffer getStatistics

Calculates the average, standard deviation and average deviation of the latencies of the captured frames in the capture buffer retrieved by **captureBuffer get**.

captureBuffer import* *fileName* *chasID* *cardID* *portID

Imports a file into the capture buffer indicated by *chasID* *cardID* *portID* from the file indicated in *fileName*; *fileName* may include a full or relative path. The format of the file is dictated by the extension on the file:

Table 3: File Extension Type

.cap	a binary format for use with the <i>captureBuffer import</i> function or IxExplorer's File Import function.
.enc	a binary format for use with NAI's Sniffer program. Note that when working with POS ports, the export function maps the POS frames to look like Ethernet data: the POS header is stripped off (4 bytes), the MAC address is padded out to 12 bytes with zeroes, a packet type identifier of 0x0800 (2 bytes, Ethernet) is added and the beginning of the MAC DA is overwritten with the POS header.

captureBuffer setConstraint

Sets the constraints used to calculate the average, standard deviation and average deviation of the latencies of the captured frames in the capture buffer retrieved by *captureBuffer get*.

captureBuffer setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *capture*.

SEE ALSO

capture

NAME - card

card - get version and type of card.

SYNOPSIS

card *sub-command options*

DESCRIPTION

This command allows the user to view version and type information for the card.

STANDARD OPTIONS

cardOperationMode

Read-only. Use the sub-command *writeOperationMode* to set this mode (ASM XMV12X card only). Options include:

Option	Value	Usage
cardOperationModeNormal		ASM XMV12X card, normal mode
cardOperationMode1geAggregated		ASM XMV12X card, 1GbE aggregated mode
cardOperationMode10geAggregated		ASM XMV12X card, 10GbE aggregated mode

cardFeatures

Read-only. The card feature command options include:

Option
cardFeatureFrequencyOffset
cardFeature1GEAggregate
cardFeature10GEAggregate
cardFeatureClockMode
cardFeaturePortClusters

clockRxRisingEdge

For 10/100 RMII cards, received data is to be clocked on the rising edge.
(*default = 1*)

clockSelect

For LSM10GXM8 cards, the currently selected clock. Options include:

Option	Value
cardClockInternal	0
cardClockExternal	1

Other options include::

Option
portGroup1Speed
portGroup2Speed
portGroup3Speed
portGroup4Speed

clockTxRisingEdge

For 10/100 RMII cards, xmit data is to be clocked on the rising edge. (*default=1*)

fpgaVersion

Read-only. The current version of central FPGA image file on this card.

hwVersion	<i>Read-only.</i> The current hardware version of this card.
portCount	<i>Read-only.</i> Number of ports on this card; if no card present, returns 0.
serialNumber	<i>Read-only.</i> For load modules which possess a serial number, this is the serial number associated with the load module.
txFrequencyDeviation	For 10GE LSM XM (NGY) and LM 10/1000/1000 TXS4 cards: a frequency deviation to be applied to the transmit clock. Values are in parts per million and vary between -102 and 102. (<i>default = 0</i>)
type	<i>Read-only.</i> The type of the card selected. The following options are used, along with the name of the card found when using IxExplorer. The Ixia part number associated with each card can be found in the <i>Ixia Hardware Guide</i> .

Option	Value	Usage	IxExplorer Name
cardNone	0	No card present	
card101004port	2	4 port 10/100 card	10/100
cardGigabit2Port	3	2 port gigabit card	Gigabit
card10100Mii	4	10/100 MII card	10/100 MII
card10100RMii	5	10/100 Reduced MII card	10/100 Reduced MII
card100FxMultiMode	6	10/100 FX multi-mode card	100Base FX MultiMode
cardGbic	7	2 port GBIC card	GBIC
cardPOS2Port	8	2 port POS card - OC12c/OC3c	OC12c/OC3c POS
cardPosOc48	9	1 port POS card	OC48c POS OC48c POS SRP/RPR/DCC
card10100Level3	10	4 port 10/100 level 3 card	10/100-3
cardGigabitLevel3	11	2 port gigabit level 3 card	Gigabit-3
cardGbicLevel3	12	2 port level GBIC card	GBIC-3
cardGigCopper	13	2 port gigabit over copper card	Gigabit
cardGigCopperLevel3	14	2 port level 3 gigabit over copper card	Gigabit-3
cardPosOc48Level3	18	1 port POS level 3 card	OC48c POS-M
cardPosOc192Plm2	22	2 port POS OC 192 Fiber Optic Board	OC192c POS
cardPosOc192Plm1	23	1 port POS OC 192 Fiber Optic Board	OC192c POS
card100FxSingleMode	26	4 port 100 FX single-mode card	100Base FX SingleMode
cardPosOc48VariableClocking	27	1 port POS card, variable clocking support	OC48c POS VAR
cardGigCopperTripleSpeed	28	2 port 10/100/1000 Copper card	Copper 10/100/1000
cardGigSingleMode	29	2 port 1000 SX Single-mode	Gigabit Single Mode
cardOc48Bert	36	1 port OC 48 card, Bit Error Rate Testing Only	OC48c POS BERT
cardOc48PosAndBert	37	1 port OC 48 card, POS and Bit Error Rate Testing	OC48c POS POS/BERT

Option	Value	Usage	IxExplorer Name
card10GEWAN2	38	2 port 10 Gigabit WAN card	OC192c POS OC192c POS/BERT OC192c VSR POS OC192c VSR POS/BERT 10GE WAN OC192c POS/BERT/10GE WAN SRP/ RPR/DCC 10GE BERT/WAN
card10GEWAN1	39	1 port 10 Gigabit WAN card	As in card10GigWanPlm2
card10GEXAUI1	46	1 port 10 Gigabit LAN XUAI card	10GE XAUI 10GE XAUI/BERT 10GE XAUI BERT
card10GigLanXenpak1	50	1 port 10 Gigabit Xenpak card	10GE XENPAK 10GE XENPAK-M 10GE XENPAK/BERT 10GE XENPAK BERT 10GE XENPAK-MA/BERT
card10GigLanXenpak2	51	2 port 10 Gigabit Xenpak card	As in card10GigLanXenpak1
card10100Txs8	57	8 port 10/100 card	10/100 TX8 10/100 TXS8
card10GELAN1	61	1 port 10 Gigabit LAN XSBI card	10GE LAN 10GE LAN-M
card10100Tx2	67	2 port 10/100 card	10/100 TX2
cardGbicSp	68	1 port GBIC card	GBIC-P1
card1000Sfps4	69	4 port 1000 SFP interface card	1000 SFP4 1000 SFPS4 1000 SFPS4-L
card1000Txs4	70	4 port 10/100/1000	10/100/1000 TX4 10/100/1000 TXS4 10/100/1000 TXS4-L
cardSingleRateBertUnframed	71	8 port singe rate unframed bert	Single-Rate Unframed BERT
cardMultiRateBertUnframed	72	8 port multiple rate unframed bert	Multi-Rate Unframed BERT
card10GEUniphy_MA	73	10GE/OC192 programmable PHY in manufacturing mode	10G UNIPHY-MA
card10GEUniphy	74	10GE/OC192 programmable PHY	10G UNIPHY
cardOc12Pos32Mb	75	Same as cardPOS2Port, but with 32MB of memory.	OC12c POS 32MB
card40GigBertUnframed	76	40GB Unframed Bert	40Gig Bert Unframed
cardOc12Atm	77	OC12 ATM	ATM 622 Multi-Rate-256MB
card1000Txs24	79	24-port 10/100/1000 for use in Optixia	10/100/1000 TX24 10/100/1000 STXS24
cardELM1000ST2	81	ELM1000ST2	10/100/100 ELM ST2
cardALM1000T8	83	ALM1000T8	10/100/100 ALM T8
card10GEXenpakP	84	1 port 10GE with Xenpak interface and enhanced processor	10GE XENPAK-P
card1000Stxs4	85	Same as card1000Txs4, but with dual-phy mode (copper and fiber)	10/100/1000 STXS4
card1000Stxs2	86	Same as card1000Stxs4, but with 2 ports	10/100/1000 STXS2
card1000Stxs1	87	Same as card1000Stxs4, but with 1 port	10/100/1000 STXS1

Option	Value	Usage	IxExplorer Name
card10GUniphyP	89	Same as 10GEUniphy, but with enhanced processor	10G UNIPHY-P
card10GELSM	90	10GE LSM	10GE LSM 10GE LSM L2/L3
card10GEMultiMSA	91	10GE Multi-MSA.	10GE Ethernet Multi-MSA 10GE Ethernet/BERT Multi-MSA 10GE Ethernet Multi-MSA-M 10GE BERT Multi-MSA 10GE Ethernet/BERT Multi-MSA-M
card10GUniphyXFP	92	Same as 10GEUniphy, but with XFP interface.	10G UNIPHY-XFP
cardPowerOverEthernet	93	4 port Power over Ethernet card	Power Over Ethernet
card2.5MSM	95	2.5 MSM (POS OC48)	2.5 MSM
cardMSM10GE	96	10G MSM	10G MSM
card10GELSMXL6	98	10GE LSM 6 ports	LSM 10GEXL6-01
cardAFMStreamExtraction Module	104	AFM1000SP-01	AFM Stream Extraction Module
card10GELSMXM3	105	LSM10GXM3	10 GE XM3
card1000XM12	106	LSM1000XMS12-01	10/100/1000 XMS12
cardLSMXMV16	107	LSM1000XMV16-01	10/100/1000 XMV16
cardASMXMV12X			10/100/1000 ASM XMV12X
card10GELSMMMacSec	112	LSM10GMS-01 (MacSec)	10GE LSM MACSec
cardLSMXMVR16	113	16 port 256MB	10/100/1000 XMVR16
cardLSMXMVR16-02	114	16 port 2GB	10/100/1000 XMV16
cardLSMXMVR12	115	12 port 1GB	10/100/1000 XMV12
cardLSMXMVR12	116	12 port 256MB	10/100/1000 XMVR12
cardLSMXMV8	117	8 port 1GB	10/100/1000 XMV8
cardLSMXMVR8	118	8 port 256MB	10/100/1000 XMVR8
cardLSMXMV4	119	4 port 1GB	10/100/1000 XMV4
cardLSMXMVR4	120	4 port 256MB	10/100/1000 XMVR4
card10GELSMMMacSec	112	MACSec	10GE LSM MACSec
card10GELSMXM8	121	NGY 8 port	10GE LSM XM8
cardVoiceQualityResourceModule	122	VQM01XM	Voice Quality Resource Module
card40GE100GELSM	123	100 GB Ethernet	100GE LSM XMV
card10GELSMXM4	124	NGY 4 port	10GE LSM XM4
card10GELSMXMR8	126	NGY 8 port reduced features	10GE LSM XMR8
card10GELSMXMR4	127	NGY 4 port reduced features	10GE LSM XMR4
card10GELSMXMR2	128	NGY 2 port reduced features	10GE LSM XMR2
card10GELSMXM8XP	129	NGY 8 port extra performance	10GE LSM XM8XP
card10GELSMXM4XP	130	NGY 4 port extra performance	10GE LSM XM4XP
card10GELSMXM2XP	131	NGY 2 port extra performance	10GE LSM XM2XP
card10GELSMXM8S	137	NGY 8 port with SFP Plus	10GE LSM XM8S
card10GELSMXM4S	138	NGY 4 port with SFP Plus	10GE LSM XM4S
card10GELSMXM2S	139	NGY 2 port with SFP Plus	10GE LSM XM2S

Option	Value	Usage	IxExplorer Name
card10GELSMXMR8S	140	NGY 8 port reduced feature with SFP Plus	10GE LSM XMR8S
card10GELSMXMR4S	141	NGY 4 port reduced feature with SFP Plus	10GE LSM XMR4S
card10GELSMXMR2S	142	NGY 2 port reduced feature with SFP Plus	10GE LSM XMR2S
cardNGYNP8	149	NGYNP 8 port with sfp plus transceiver	NGY-NP8
cardNGYNP4	150	NGYNP 4 port with sfp plus transceiver	NGY-NP4
cardNGYNP2	151	NGYNP 2 port with sfp plus transceiver	NGY-NP2
cardFCMGXM8	154	8 port FCM SFP+ card	FCM GXM8
cardFCMGXM4	168	4 port FCM SFP+ card	FCM GXM4
cardXDM10G32S	170	32 port Ethernet SFP+ card	XDM10G32S
cardFlexAP10G16S	161	Excellon-Flex 16 port full performance 10G card	FlexAP10G16S
cardFlexFE10G16S	171	Excellon-Flex 16 port reduced version 10G card	FlexFE10G16S
cardEthernetVMS	173	For single NIC Ethernet VM ports	
cardEthernetVM	152	For multi NIC Ethernet VM ports	IxVM L2L3 Multi Port (or IxVM L4L7 Multi Port)

typeName

Read-only. The name corresponding to the card type. One of the symbolic values shown under *type*.

DEPRECATED OPTIONS
type

The following *card type* options have been deprecated:

Option	Value	Usage
cardUSB	19	4 port 10 Mbps/USB card

txClockDeviationLan

For 10GE LSM XM8 cards. LAN transmit clock deviation in units of ppm, referred to as Frequency Offset in IxHal.

txClockDeviationWan

For 10GE LSM XM8 cards. WAN transmit clock deviation in units of ppm.

clockType

The following options have been deprecated:

Option	Value	Usage
cardBertUnframedClockSonet	0	(default) 155.52 Mbps (OC-3), 622.08 Mbps (OC-12) and 2.488 Gbps (OC-48) data rates
cardBertUnframedClockSonetWithFEC	1	166.63 Mbps (OC-3 FEC), 666.51 Mbps (OC-12 FEC) and 2.67 Gbps (OC-48 FEC) data rates
cardBertUnframedClockFiberChannel	2	1.062 Gbps (Fibre Channel) and 2.124 Gbps (2x Fibre Channel) data rates
cardBertUnframedClockGigE	3	1.25Gbps (Gigabit Ethernet) data rates

Option	Value	Usage
cardBertUnframedClockExternal	4	Clock is externally supplied.

COMMANDS

The **card** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

card config *option value*

Modify the configuration options of the card. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for card.

card cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **card** command.

card export *fileName chasID cardID*

Exports the current configuration of the card at slot *cardID*, chassis *chasID* into the file named *fileName*; *fileName* may include a full or relative path. If no extension is used in *fileName*, a “.crd” is added to the name. The file produced by this command may be used by the *import* sub-command. Specific errors are:

- No connection to a chassis
- Invalid card

card get *chasID cardID*

Gets the current configuration of the card at slot *cardID*, chassis *chasID*. Call this command before calling **card cget *option value*** to get the value of the configuration option. If the card does not exist, an error is returned.

card getPortClusterCount *chasID cardID*

Gets the port cluster count.

card getFrontPanelMasterPort *chasID cardID portClusterIndex*

Gets the front panel master port.

card getCpuPortList *chasID cardID portClusterIndex*

Gets the CPU Port list.

card getFrontPanelPortList *chasID cardID portClusterIndex*

Gets the front panel port list.

card getPortClusterIndex *chasID cardID portID*

Gets the index of the port cluster.

card getAssociatedFrontPanelPorts *chasID cardID cpuPortID*

Gets the associated front panel ports.

card import *fileName chasID cardID*

Imports a saved card configuration found in the file *fileName* into the current configuration of the card at slot *cardID*, chassis *chasID*. *fileName* may include a full or relative path. If no extension is used in *fileName*, a “.crd” is added to the name. The file used by this command must have been produced by the *export* sub-command.

Note: This command should be followed by *ixWritePortsToHardware* to write the stream and protocol configurations to hardware, and *card write* to write the card parameters to hardware.

Specific errors are:

- No connection to a chassis
- Invalid card
- The card is owned by another user
- *fileName* does not exist

card *isActiveFeature* *chasID* *cardID* *feature*

Determines whether a specific feature is active for the card at *cardID*, chassis *chasID* and that the card is properly configured/enabled to use that feature. A value of true (1) is returned if the feature is enabled and false (0) if the feature is not enabled. *Feature* may be one of the values from the *isValidFeatures* list.

card *isCpuPort* *chasID* *cardID* *portID*

Determines whether Port CPU is active.

card *isFrontPanelMasterPort* *chasID* *cardID* *portID*

Determines whether front panel of master port is active.

card *isValidFeature* *chasID* *cardID* *feature*

Determines whether a specific feature is valid for the card at *cardID*, chassis *chasID* with the card in its current mode. A value of true (1) is returned if the feature is valid and false (0) if the feature is invalid or the port is invalid. *Feature* may be one of the following values:

Option	Value	Usage
cardFeatureFrequencyOffset	131	Frequency offset
cardFeature1GEAggregate	280	ASM XMV12X card, 1GbE aggregated mode
cardFeature10GEAggregate	281	ASM XMV12X card, 10GbE aggregated mode
cardFeatureClockMode	313	Clock mode
cardFeaturePortClusters	432	Port clusters feature

card *set* *chasID* *cardID*

Sets the current configuration of the card at slot *cardID*, chassis *chasID* by reading the configuration option values set by the **card config option value** command. Specific errors are:

- No connection to a chassis
- Invalid parameters
- Network problem between the client and chassis

card *setDefault*

Sets to IxTclHal default values for all configuration options.

card *setFactoryDefaults* *chasID* *cardID*

Sets factory default information. The specific card write sends only card specific information. Set "card write *chasID* *cardID*" after setting card *setFactoryDefaults* to get specific information.

card write *chasID cardID*

Writes the card properties of the card at slot *cardID*, chassis *chasID* by reading the configuration option values set by the **card config *option value*** command. No *stream* or *port* properties are written.

card writeOperationMode *mode chasID cardID*

Writes the mode for the card. See the option *cardOperationMode*. Note: This also writes the port configuration for all ports on the card (ASM XMV12X card only).

card addVMPort *chasID cardID portID testNic promiscuous_mode testMac linkMTU lineSpeed*

Adds IxVM port on a virtual appliance.

The return values for the command are:

Options	Values
kPortOK	0
kPortIdExists	10
kPortIdOutOfRange	11
kPortOverlappingNIC	12
kPortIdUnavailable	13
kPortBuildTimeout	14
kPortInvalidOnSingleNic	15
kPortRemoveTimeout	17

The parameters for the commands are:

Option	Value	Usage
<i>keepAliveTimeout</i>	any value in seconds (>= 3)	Indicates that after the buffer is full, no more frames are to be captured.
<i>testNic</i>		The existing nic name of a virtual machine, used to generate traffic (port), Example: eth1.
<i>promiscuous_mode</i>	0/1	Enables promiscuous mode on port (Promiscuous mode must be supported and enabled in hypervisor also, for this option to work).
<i>testMac</i>		The MAC address of the port. This must match the MAC address of the interface with the name provided in <i>testNic</i> param, can be added as 00:00:00:00:00:00, and is automatically taken from interface at add time.
<i>linkMTU</i>	default = 1500	A valid Ethernet MTU.

Option	Value	Usage
<i>lineSpeed</i>	<i>valid values: 100Mbps and 1000Mbps (new versions of IxOS limits the speed to this 2 values, using IxN and IxL user can also set the speed up to 10000 Mbps, in increment of 1k)</i>	The virtual line speed at which the port will be capped, in software. Default = 1000. If an invalid value is entered speed will be set to 1000.

card **createPartition** *chasID cardID phyPortList cpuPortList*

Creates partition in port list CPU.

card **queryPartition** *chasID cardID partitionID*

Sends query for partition.

card **deletePartition** *chasID cardID partitionID*

Deletes partition in port list CPU.

card **removeVMPort** *chasID cardID portID*

Removes IxVM port on a virtual appliance.

card **resetHardware** *chasID cardID*

With this command, the chassis resets all the hardware, reboots port CPU, tests the local processor test, and rewrites the streams. This command does not modify existing port/stream configuration.

card **getMaxResourceGroups** *chasID cardID*

With this command, gets the maximum number of supported resource groups.

card **addResourceGroup** *chasID cardID*

Adds resource groups.

card **createResourceGroups** *chasID cardID*

Creates resource groups.

card **deleteResourceGroups** *chasID cardID*

Deletes resource groups.

card **getConfiguredResourceGroupList** *chasID cardID*

Gets and configures resource group list.

card **setConfiguredResourceGroupList** *chasID cardID*

Sets and configures resource group list.

card **forceHotswap** *chassisID cardID*

Deliberately forces hotswap of the card.

Example:

```
package require IxTclHal
ixConnectToChassis $ChassisId
TclScripts) 2 % card forceHotswap 1 2
```

Switch Mode

The following commands enable the Xcellon-Multis card to switch mode:

For CXP Module:

```
card get 1 2
resourceGroupEx get 1 2 1( it can be 1 2 5, 1 2 6, and 1 2 7)
resourceGroupEx cget -activePortList #it displays current mode
resourceGroupEx config -activePortList {0 2 1} #for 100G mode
resourceGroupEx config -activePortList {{0 2 5}{0 2 6}{0 2 7}}
#for 40/10G mode

resourceGroupEx config -mode 8 #for 40G mode
resourceGroupEx config -mode 10 #for 10G mode

resourceGroupEx set 1 2 1( it can be 1 2 5, 1 2 6, and 1 2 7)
resourceGroupEx write 1 2 1 ( it will send message to and execute
mode switch )
```

For QSFP Module:

```
card get 1 2
resourceGroupEx get 1 2 1
resourceGroupEx cget -activePortList #it displays current mode
resourceGroupEx config -activePortList {{0 2 1}{0 2 2}{0 2 3}}
#for 40/10G mode

resourceGroupEx config -mode 8 #for 40G mode
resourceGroupEx config -mode 10 #for 10G mode

resourceGroupEx set 1 2 1
resourceGroupEx write 1 2 1 ( it will send message to and execute
mode switch )
```

Capture Playback

Capture Playback is a resource group mode on the Xcellon Multis load module that allows you to load a packet capture file into port hardware. Once the capture file is successfully loaded, you will be able to transmit all of the loaded packets.

Capture Playback supports the .pcap, .pcapng, and .enc capture file formats.

A sample workflow is provided as follows:

```
# setup some basic variables
set chassisName "user-chassis"
set chassID 1
set cardID 1
set mode40G_CPB 12
set mode100G_CPB 11
set continuousPackets 0
set burstPackets 1
set loopPackets 5
set capFile "sample.pcap"

# connect to the chassis
ixConnectToChassis $chassisName

# switch ports 1 and 2 into 100 Capture Playback mode
# NOTE: the mode switch process can take a substantial
# amount of time so prepare to wait at least several
# minutes for it to complete.

resourceGroupEx get $chassID $cardID 1
resourceGroupEx config -mode $mode100G_CPB
resourceGroupEx set $chassID $cardID 1
```

```

resourceGroupEx write $chassID $cardID 1

resourceGroupEx get $chassID $cardID 2
resourceGroupEx config -mode $mode100G_CPB
resourceGroupEx set $chassID $cardID 2
resourceGroupEx write $chassID $cardID 2

# create a port group for ports 1 and 2
portGroup destroy 1
portGroup create 1
portGroup add 1 $chassID $cardID 1
portGroup add 1 $chassID $cardID 2
portGroup write 1

# configure capture playback for each port and set the
# transmit mode to burst mode.
# NOTE: that this is the configuration step for capture
# playback and it must come BEFORE the loading of the
# packet file.

capturePlayback get $chassID $cardID 1
capturePlayback config -framesPerSec    30
capturePlayback config -framesToBurst   9
capturePlayback config -transmitType    $burstPackets
capturePlayback config -captureFileChassis $capFile
capturePlayback set $chassID $cardID 1
capturePlayback write $chassID $cardID 1

capturePlayback get $chassID $cardID 2
capturePlayback config -framesPerSec    30
capturePlayback config -framesToBurst   9
capturePlayback config -transmitType    $burstPackets
capturePlayback config -captureFileChassis $capFile
capturePlayback set $chassID $cardID 2
capturePlayback write $chassID $cardID 2

# load the packet file for each port
capturePlayback load $chassID $cardID 1
capturePlayback load $chassID $cardID 2

# the port group that was created prior to configuring
# capture playback can now be used to send a burst
# of packets.
portGroup setCommand 1 $::sendNextBurstCP

# individual ports can be made to send a burst of packets
capturePlayback sendNextBurst $chassID $cardID 1
capturePlayback sendNextBurst $chassID $cardID 2

```

DEPRECATED COMMANDS

card getInterface *chassisID cardID*

Gets the interface type of the card.

EXAMPLES

```
package require IxTclHal
```

```
set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis' number of cards
chassis getFromID $chas
set ncards [chassis cget -maxCardCount]
ixPuts "Chassis $chas, $ncards cards"

for {set i 1} {$i <= $ncards} {incr i} {
    # Check for missing card
    if {[card get $chas $i] != 0} {
        continue
    }
    set portList [list [list $chas $i 1]]
    # Take ownership of the ports we'll use
    if [ixTakeOwnership $portList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Get all of the card's characteristics
    set fpgaVersion [card cget -fpgaVersion]
    set hwVersion   [card cget -hwVersion]
    set portCount  [card cget -portCount]
    set type       [card cget -type]
    set typeName   [card cget -typeName]

    # And list them
    ixPuts "Card $i: $typeName ($type), $portCount ports, \
            fpga $fpgaVersion, hwVersion $hwVersion"

    # If the card is a 10/100 RMII, play with its settable
    parameters
    if {$type == $::card10100RMii} {
        card config -clockRxRisingEdge 0
        card config -clockTxRisingEdge 1
        if [card set $chas $i] {
            ixPuts "Could not card set $chas $i"
```

```

        }
        ixWriteConfigToHardware portList
    }

    # Just for fun, we'll export the data associated with the first
    card
    # and read it to any other cards of the same type
    if {$i == 1} {
        if {[card export cardfile $chassis $i] != 0} {
            ixPuts "Could not export"
        } else {
            set savedType $type
        }
    } elseif {$type == $savedType} {
        if {[card import cardfile $chassis $i] == 1} {
            ixPuts "Could not import"
        }
    }
    # Let go of the ports that we reserved
    ixClearOwnership $portList
}

# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
# Everest 10/40G Combo (assume chassis = 1; card = 6):
# Example to configure 2 resource groups in 40G mode and
# 1 resource group in 10G aggregation mode

# Check if the card supports resource group -
kFeatureResourceGroup
set validRG [card isValidFeature 1 6 454]

# Check if the card is Everest 10/40G Combo -
kFeatureEverest40GCombo.
set validEverestCombo [card isValidFeature 1 6 435]

# Get the max number of supported resource groups
set maxRG [card getMaxResourceGroups 1 6]

# Check if the card is Everest 10/40G Combo. Please note we have
# 5 ports in resource group for 40G combo, while we have 4 ports in
# resource group for Everest 10G
card get 1 6
set numPorts [card cget -portCount] # 20 for combo; 16 for
Everest 10G
set portsPerRG [expr $numPorts / $maxRG]

# step1: Add ports in resource group
# Set {1, 2, 3, 4, 17} and {9, 10, 11, 12, 19} to 40G mode
# Set {5, 6, 7, 8, 18} to 10G aggregation mode
set rgPorts1 [list [list 1] [list 2] [list 3] [list 4] [list 17]]
set rgPorts2 [list [list 5] [list 6] [list 7] [list 8] [list 18]]
set rgPorts3 [list [list 9] [list 10] [list 11] [list 12] [list
19]]

card addResourceGroup 17 40000 $rgPorts1 # active port 17, speed
40000

```

```

card addResourceGroup 19 40000 $rgPorts3 # active port 19, speed
40000
card addResourceGroup 5 10000 $rgPorts2 # active port 5, speed
10000
# step2: Create Resource group with the configured groups
# This call will push down the configuration in the server
# This is a blocking call and will fail if any of the groups are
# configured incorrectly or all the ports are not owned etc.
card createResourceGroups 1 6

# Get the configured Resource Groups list
set configuredRGList [card getConfiguredResourceGroupList 1 6]
return value: {17 40000 {1 2 3 4 17}} {5 10000 {5 6 7 8 18}} {19
40000 {9 10 11 12 19} }

# Set the configured Resource Groups list
# This api can be directly used to set one or more resource
# groups in a card.
# Note: setConfiguredResourceGroupList can be directly fed the
output
# from getConfiguredResourceGroupList
set configuredRGList [card getConfiguredResourceGroupList 1 6]
card setConfiguredResourceGroupList 1 6 $configuredRGList

card setConfiguredResourceGroupList $chassis $card {{17 40000 {1 2
3 4 17}} {18 40000 {5 6 7 8 18}} {19 40000 {9 10 11 12 19}} {20
40000 {13 14 15 16 20}}}

# Delete Resource Groups. This will put the ports in the RG in
# normal 10G mode.
set dList [list [list 17] [list 19] [list 9]]
card deleteResourceGroups 1 6 $dList

***Note: For Everest 10G only speed 10000 is supported and
resource group contains 4 ports. All other steps are exactly same.
Please look at the example below:
# step1: Add ports in resource group
# Set {1, 2, 3, 4} and {9, 10, 11, 12} to 10G aggregation mode
set rgPorts1 [list [list 1] [list 2] [list 3] [list 4]]
set rgPorts3 [list [list 9] [list 10] [list 11] [list 12]]

card addResourceGroup 1 10000 $rgPorts1 # active port 1, speed
10000
card addResourceGroup 5 10000 $rgPorts3 # active port 5, speed
10000

```

SEE ALSO*chassis, port*

NAME - resourceGroupEx

resourceGroupEx- get version and type of resource group.

SYNOPSIS	resourceGroupEx <i>sub-command options</i>
DESCRIPTION	This command allows the user to view version and type information for the resource group ex.
COMMANDS	The resourceGroupEx command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
resourceGroupEx config <i>option value</i>	Modify the configuration options of the resource group ex.
resourceGroupEx cget <i>option</i>	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the resource group ex command.
resourceGroupEx cget activePortList	To query active port list so to identify the current speed mode.
resourceGroupEx config activePortList	To select active port list for either 100G or 40G or 10G to perform speed mode switch.
resourceGroupEx cget capturePortList	Query current capture port list.
resourceGroupEx config capturePortList	To select capture port list. For XM100GE4CXP+FAN+10G , it selects one port out of 3 ports (one per resource group) as active capture port. In past, all ports have capture capability. But with XM100GE4CXP+FAN+10G under 40G/10G mode, only one port at a time can be the active capture port from one port group. They are (5, 6,7) (8,9,10) (11,12,13) (14,15,16). For XM10/40GE06QSFP+FAN+10GE , it selects one port out of 3 ports (one per resource group)as active capture port. In past, all ports have capture capability. But with XM10/40GE06QSFP+FAN+10GE under 40G/10G mode, only one port at a time can be the active capture port from one port group. They are (1, 2,3) (4,5,6). For XM10/40GE12QSFP+FAN+10GE , it selects one port out of 3 ports (one per resource group)as active capture port. In past, all ports have capture capability. But with XM10/40GE12QSFP+FAN+10GE under 40G/10G mode, only one port at a time can be the active capture port from one port group. They are (1, 2,3) (4,5,6) (7, 8,9) (10,11,12).

resourceGroupEx **config mode**

Switches speed configuration per resource group.

Table A-1. Modes and their corresponding speeds

Mode	Speed	Interface
7	1*100G	CXP
8	3*40G	CXP
9	8*10G	CXP
10	3*10G	CXP
8	3*40G	QSFP
9	8*10G	QSFP
10	3*10G	QSFP

resourceGroupEx **cget ppm**

Query current ppm.

resourceGroupEx **cget ppm 100**

To set current ppm.

PPM can be set on port group level instead of card or port level. PPM is the frequency offset. It is the transmit clock deviation expressed in parts per million (ppm). The default is 0.= 0). It can be set from -100 to 100.

resourceGroupEx **get chasID cardID groupID**

It gets configuration to hardware.

resourceGroupEx **set chasID cardID**

It sets configuration to hardware.

resourceGroupEx **write chasID cardID groupID**

It writes configuration to hardware.

NAME - cdlPreamble

cdlPreamble - configure the transmit CDL preamble

SYNOPSIS	cdlPreamble <i>sub-command options</i>
DESCRIPTION	The <i>cdlPreamble</i> command is used to set the CDL preamble values when <i>txRxPreamble enableCiscoCDL</i> is set to <i>true</i> . It is also used to receive the decoded value from a captured frame.
STANDARD OPTIONS	
applicationSpecific	Four bytes of application specific data. For example, “0x11223344”. (<i>default = “55 55 555 55”</i>)
cdlHeader	<i>Read-only</i> . The resultant combined CDL header, as a hex list. For example, “55 55 55 55 55 D5”.
enableHeaderCrc Overwrite true / false	If <i>true</i> , then the value in <i>headerCrc</i> is used to overwrite the calculated value of the header CRC in the CDL preamble. (<i>default = true</i>)
headerCrc	If <i>enableHeaderCrcOverwrite</i> is <i>true</i> , then this value is used to replace the automatically calculated CRC.
messageChannel	The in-band message channel, a one byte quantity. (<i>default = 0x55</i>)
oam	The packet type and OAM field, a one byte quantity. (<i>default = 0x55</i>)
startOfFrame	<i>Read-only</i> . The Start of Frame indicator, always 0xFB.
COMMANDS	
	The <i>cdlPreamble</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
cdlPreamble cget option Returns the current value of the configuration option <i>cdlPreamble</i> by <i>option</i> . <i>Option</i> may have any of the values accepted by the <i>cdlPreamble</i> command, subject to the setting of the <i>enableValidStats</i> option.	
cdlPreamble config option value Modify the configuration options of the time server. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for <i>cdlPreamble</i> .	
cdlPreamble decode capFrame chasID cardID portID Decodes a captured frame in the capture buffer and makes the values of the decoded header available in the options of this command. Specific errors are:	
<ul style="list-style-type: none"> • No connection to a chassis • The captured frame is not a valid CDL packet 	

cdlPreamble get* *chasID cardID portID

Gets the current preamble configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Call this command before calling *cdlPreamble cget option* to get the value of the configuration option.

cdlPreamble set* *chasID cardID portID

Sets the preamble configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the *cdlPreamble config option value* command.

cdlPreamble setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
set chasID 1
set cardID 4
set portID 1

txRxPreamble setDefault
txRxPreamble config -txMode      preambleByteCount
txRxPreamble config -rxMode      preambleSameAsTransmit

if [port isValidFeature $chasID $cardID $portID /
portFeatureCiscoCDL]

{
    txRxPreamble config -enableCiscoCDL true

    if [txRxPreamble set $chasID $cardID $portID] {
        ixPuts $::ixErrorInfo
    }

    cdlPreamble config -oam 55
    cdlPreamble config -applicationSpecific {11 22}
    if [cdlPreamble set $chasID $cardID $portID] {
        ixPuts $::ixErrorInfo
    }
}
```

SEE ALSO

stream, *txRxPreamble*

NAME - cfpPort

cfpPort - configure the transmit CFP port.

SYNOPSIS	cfpPort <i>sub-command options</i>
DESCRIPTION	The <i>cfpPort</i> command is used to set the CFP port values.
<hr/>	
STANDARD OPTIONS	
getIxiaCfpAdapterType	. The cfpPort getIxiaCfpAdapter command returns the type of ixia cfp adapter currently plugged in. There are 3 different values returned: <ul style="list-style-type: none"> • 0 (None) – there isn't an ixia adapter plugged in. It's a cfp from another vendor, or no cfp. • 1 (QSFP) – there is an ixia adapter plugged in with a QSFP interface (it could be dual or single). • 2 (CXP) - these is an ixia adapter plugged in with a CXP interface.
getModuleId	Gets module identifier.
getVendor	Gets the vendor for CFP.
isIxiaCfpAdapter	Signifies if Ixia gets CFP adapter.
enableDualPortOperati on	If true, enables dual port operation.
transmitClockDeviation	For ports that support the <i>portFeatureFrequencyOffset</i> feature, this is the transmit clock deviation expressed in parts per million (ppm). (<i>default = 0</i>). A 'cfpPort' in dual port mode has two 'ports' in single port mode it has one. When dual mode is enabled both ports on a dual adapter will have the same deviation. In single port mode the clock deviation is still programmed as a port property instead of a CfpPort property.
COMMANDS	The <i>cfpPort</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
cfpPort set <i>chassisId, cardId, portId</i>	Singifies the set option for <i>cfpPortID</i> .
cfpPort get <i>chassisId, cardId, portId</i>	Singifies the get option for <i>cfpPortID</i> .
cfpPort write <i>chassisId, cardId, portId</i>	Singifies the write option for <i>cfpPortID</i> .
cfpPort forceEnablePort <i>chassisId, cardId, portId</i>	Singifies when port is forcefully enabled.

A

cfpPort

cfpPort *isDualCfpPort* *chassisId*, *cardId*, *portId*

Signifies if CFP port is dual or not.

SEE ALSO

NAME - chassis

chassis - add a new chassis to the chain and configure it.

SYNOPSIS

chassis *sub-command options*

DESCRIPTION

The **chassis** command is used to add a new chassis to a chain of chassis, configure an existing chassis or delete an existing one from the chain in use.

STANDARD OPTIONS

baseIpAddress

The IP address that is used to get to the port CPUs. In IxExplorer, this is "IxRemoteIp" under Chassis Properties. (*default = 10.0.0.0*)

baseAddressMask

The mask address that is used to get the port CPUs.

cableLength

Specifies the length of the cable between all chassis. Options include:

Option	Value	Usage
cable3feet	0	default
cable6feet	1	
cable9feet	2	
cable12feet	3	
cable15feet	4	
cable18feet	5	
cable21feet	6	
cable24feet	7	

hostname

Read-only. The hostname associated with the chassis, as specified in the last *chassis add* operation.

id

ID number given to the chassis. (*default = 0*)

ipAddress

Read-only. The IP address associated with the chassis.

ixServerVersion

Read-only. The installed IxOS version associated with the chassis.

master *true/false*

Read-only. Specifies whether this chassis is a master or a slave in a chain. There can be only one master chassis in a chain. Note: The master is automatically assigned based on cable connections.

maxCardCount

Read-only. Number of card can be installed on the chassis.

name

The given name of the chassis. (*default = defaultChassis*)

operatingSystem

Read-only. The operating system loaded on the chassis.

Option	Value	Usage
chassisOSUnknown	0	Unknown operating system

Option	Value	Usage
chassisOSWin95	1	Windows 95
chassisOSWinNT	2	Windows NT
chassisOSWin2000	3	Windows 2000
chassisOSWinXP	4	Windows XP
chassisOSWin7	5	Windows 7

powerConsumption

The power consumption level of the port CPU.

sequence

Specifies the sequence number of the chassis in the chain. The master must have a sequence number of 0 and other chassis should be incrementing. (*default = 1*)

syncInOutCountStatus

Specifies the sync-in and sync-out count status.

type

Read-only. Specifies the type of chassis. Possible values are:

Option	Value	Usage
ixia1600	2	16 card chassis type
ixia200	3	2 card chassis type
ixia400	4	4 card chassis type
ixia100	5	1 card chassis type with GPS
ixia400C	6	1 card chassis with additional power and fans
ixia1600T	7	16 card chassis type with additional power and fans
ixiaDemo	9	128 card chassis type used in demo server
ixiaOptixia	10	Optixia chassis
ixiaOpixJr	11	Ixia test board
ixia400T	14	4 card chassis type
ixia250	17	2 card chassis type
ixia400Tf	18	4 card chassis type, special fan speed
ixiaOptixiaX16	19	16 card chassis type
ixiaOptixiaXL10	20	10 card chassis type
ixiaOptixiaXM12	22	12 card chassis type
ixiaOptixiaXV	24	virtual chassis (OptixiaXV)

typeName

Read-only. The printable chassis type name.

DEPRECATED OPTIONS**baseAddressMask**

This option has been deprecated (with IxOS version 5.0).

COMMANDS

The **chassis** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

chassis add sIPAddr

Adds a new chassis with *sIPAddr* (hostname or IP address) to the chain. Specific errors are:

- Error connecting to the chassis (timeout, invalid IP or hostname, or invalid port) (1)
- Version mismatch (2)
- The version was successfully negotiated, but a timeout occurred receiving the chassis configuration (3)
- Hardware conflict (4)

chassis cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **chassis** command.

chassis config option value

Modify the configuration options of the chassis. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for chassis.

chassis del sIPAddr

Deletes the chassis with *sIPAddr* (hostname or IP address) from the chain.

chassis export fileName sIPAddr

Exports the current configuration of the chassis *sIPAddr* (hostname or IP address) into the file named *fileName*; *fileName* may include a full or relative path. The file produced by this command may be used by the *import* sub-command. Specific errors are:

- No connection to a chassis

chassis get sIPAddr

Gets the current configuration of the chassis with *sIPAddr* (hostname or IP address) from hardware. Call this command before calling **chassis cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis

chassis getChassisPowerConsumption i_IPAddr

Gets the chassis power consumption value.

chassis getChassisSyncInOutCount sIPAddr

Gets the sync-in and sync-out count of the chassis with *sIPAddr* (hostname or IP address) from the hardware.

chassis getFromID chasID

Gets the current configuration of the chassis with *chasID* from hardware. Call this command before calling **chassis cget option value** to get the value of the configuration option.

chassis import fileName sIPAddr

Imports a saved chassis configuration found in the file *fileName* into the current configuration of the chassis *sIPAddr* (hostname or IP address). *fileName* may include a full or relative path. The file used by this command must have been produced by the *export* sub-command. A *chassis write* is necessary to commit

these items to the hardware. You must have chassis-wide rights to use this command. Specific errors are:

- No connection to a chassis
- Network problem between the client and chassis
- User doesn't have chassis-wide rights.

chassis reboot *chassisID*

Reboots the chassis. You must have chassis-wide rights to use this command. Specific errors are:

- No connection to a chassis
- Network problem between the client and chassis
- User doesn't have chassis-wide rights.

chassis refresh *sIPAddr*

Ensures that the data displayed is up to date. Specific errors are:

- No connection to a chassis
- Network problem between the client and chassis

chassis resetHardware *sIPAddr*

Resets the hardware by initializing all the registers and statistic counters. You must have chassis-wide rights to use this command. Specific errors are:

- No connection to a chassis
- Network problem between the client and chassis
- User doesn't have chassis-wide rights.

chassis saveIxFile *sIpAddr*

Saves the current IxServer configuration to an IxServer.iks file on the server. The IxServer.iks file is saved automatically when there is a orderly shutdown of IxServer. It may also be saved manually, by using this command, to ensure backup of the configuration without having to shut down the system, such as in service monitoring situations. If some unexpected shutdown occurs, the IxServer configuration is reloaded from the saved .iks file on power-up.

The sIpAddr should be the IP address of the current chassis.

chassis set *sIPAddr*

Sets the entire configuration of the chassis, including *baseIpAddress*, in IxHAL with *sIPAddr* (hostname or IP address of the chassis) by reading the configuration option values set by the **chassis config *option value*** command. Specific errors are:

- No connection to a chassis
- Network problem between the client and chassis
- User doesn't have chassis-wide rights.

chassis setBaseIp *sIPAddr*

Sets only the base IP address for the chassis with *sIPAddr* (hostname or IP address of the chassis). In IxExplorer Chassis Properties, this is named '*IxRemoteIp*'. You must have chassis-wide rights to use this command. Specific errors are:

- No connection to a chassis
- Network problem between the client and chassis
- User doesn't have chassis-wide rights.

chassis setDefault

Sets to IxTclHal default values for all configuration options.

chassis setFactoryDefaults *sIPAddr*

Sets the factory default values on the chassis. You must have chassis-wide rights to use this command. Specific errors are:

- No connection to a chassis
- User doesn't have chassis-wide rights.
- User doesn't have chassis-wide rights.

chassis shutdown *sIPAddr*

Shuts down the chassis. You must have chassis-wide rights to use this command. Specific errors are:

- No connection to a chassis
- Network problem between the client and chassis
- User doesn't have chassis-wide rights.

chassis addVirtualCard *chassisIP cardIP cardId keepAliveTimeout*

Adds virtual machine card to the chassis. You must have chassis-wide rights to use this command. Specific errors are:

- No connection to a chassis
- Network problem between the client and chassis
- User doesn't have chassis-wide rights.

The return values for the command are:

Options	Values
kCardOK	0
kCardIdExists	5
kCardIdOutOfRange	6
kCardOverlappingIP	7
kCardBuildTimeout	8
kCardIdUnavailable	9
kCardRemoveTimeout	16

The parameters for the commands are:

Option	Value	Usage
<i>keepAliveTimeout</i>	<i>any value in seconds (>= 3)</i>	Indicates that after the buffer is full, no more frames are to be captured.
<i>testNic</i>		The existing nic name of a virtual machine, used to generate traffic (port), Example: eth1.

Option	Value	Usage
<i>promiscuous_mode</i>	0/1	Enables promiscuous mode on port (Promiscuous mode must be supported and enabled in hypervisor also, for this option to work).
<i>testMac</i>		The MAC address of port. This must match the MAC address of the interface with the name provided in <i>testNic</i> param, can be added as 00:00:00:00:00:00, and is automatically taken from interface at add time.
<i>linkMTU</i>	default = 1500	A valid Ethernet MTU.
<i>lineSpeed</i>	<i>valid values: 100Mbps and 1000Mbps (new versions of IxOS limits the speed to this 2 values, using IxN and IxL user can also set the speed up to 10000 Mbps, in increment of 1k)</i>	The virtual line speed at which the port will be capped, in software. Default = 1000. If an invalid value is entered speed will be set to 1000.

chassis removeVMCard *chassisIP cardId*

Removes virtual machine card from the chassis. You must have chassis-wide rights to use this command. Specific errors are:

- No connection to a chassis
- Network problem between the client and chassis
- User doesn't have chassis-wide rights.

chassis checkVMForDuplicate *chassisIP cardIP*

Checks if the virtual machine already exists on the chassis. You must have chassis-wide rights to use this command. Specific errors are:

- No connection to a chassis
- Network problem between the client and chassis
- User doesn't have chassis-wide rights.

chassis forceHotswap *chassisIPAddr cardID*

Deliberately forces hotswap of the card.

Example:

```
package require IxTclHal
ixConnectToChassis $ChassisId
TclScripts) 1 % chassis forceHotswap 10.205.27.99 2
```

DEPRECATED COMMANDS

chassis write *chasID cardID portID*

Do not use.

chassis addVMCard *chassisIP cardIP cardId cardType keepAliveTimeout ixvmVCardExtType*

Do not use.

EXAMPLES

```

package require IxTclHal

# Set up two chassis in a chain
set host1 galaxy
set host2 localhost

# Remove all of the chassis in the chain
chassisChain removeAll

-----
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis [list $host1 $host2]] {
    ixPuts $::ixErrorInfo
    return 1
}

# Check for a valid chain
if [chassisChain validChain] {
    ixPuts "Chain has no master"
}

set masterSlave(0) slave
set masterSlave(1) master

# Get the type and capabilities of the chassis
chassis get $host1
set chas1 [chassis cget -id]
set type [chassis cget -type]

ixPuts -nonewline "Chassis $host1 (id $chas1) is type: "
switch $type \
    $::ixia1600      {ixPuts -nonewline "IXIA 1600"} \
    $::ixia200      {ixPuts -nonewline "IXIA 200"} \
    $::ixia400      {ixPuts -nonewline "IXIA 400"} \
    $::ixia100      {ixPuts -nonewline "IXIA 100"} \
    $::ixia400C     {ixPuts -nonewline "IXIA 400C"} \
    $::ixial600T    {ixPuts -nonewline "IXIA 1600T"} \
    $::ixiaDemo     {ixPuts -nonewline "IXIA Demo"} \
    $::ixiaOptIxia  {ixPuts -nonewline "IXIA OptIxia"} \
    $::ixiaOpixJr   {ixPuts -nonewline "IXIA OpixJr"} \
    default         {ixPuts -nonewline "Unknown"} \


set maxCards [chassis cget -maxCardCount]
ixPuts ", which can accommodate $maxCards cards"

chassisChain removeAll
# Add a chassis as the master
chassis setDefault
chassis config -id 1
chassis config -sequence 1
chassis add $host1

# And give it a name after the fact

```

```
chassis config -name "test-chassis"
chassis set $host1

# Make sure it's the master
chassis getFromID 1
set master [chassis cget -master]
ixPuts "$host1 is $masterSlave($master)"

chassis setDefault
chassis config -id 2
chassis config -sequence 2
chassis config -cableLength cable6feet
chassis add $host2

# Make sure it's not the master
chassis getFromID 2
set master [chassis cget -master]
ixPuts "$host2 is $masterSlave($master)"

# Release the chassis
chassis del $host1
chassis del $host2

# Disconnect from the chassis we're using
ixDisconnectFromChassis [list $host1 $host2]
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

NAME - chassisChain

chassisChain - configure an entire chassis chain

SYNOPSIS	chassisChain <i>sub-command options</i>
DESCRIPTION	The chassisChain command is used to write configuration parameters to all chassis in the chain.
STANDARD OPTIONS	
delayChassisStartTime	The number of seconds to delay test application after a start.(<i>default = 5</i>)
COMMANDS	
The chassisChain command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.	
chassisChain broadcastTopology After a chassis is added to or deleted from a chain, it must broadcast its existence to the rest of the chassis in the chain. Note: This command doesn't return a value.	
chassisChain cget option Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the chassisChain command.	
chassisChain config option value Modify the configuration options of the chassisChain. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for chassisChain.	
chassisChain get Gets the current configuration of the chassisChain. Call this command before calling chassisChain cget option value to get the value of the configuration option.	
chassisChain set Sets the Chassis Chain configuration by reading the configuration option values set by the chassisChain config option value command.	
chassisChain setDefault Sets to IxTclHal default values for all configuration options.	
chassisChain removeAll Removes or disconnects from all the chassis in the chain.	

chassisChain validChain

Verify whether the chain is valid. A valid chain has at least one chassis assigned as a master. Specific errors are:

- There is no master in the chain

chassisChain write *groupID*

Writes or commits the changes in IxHAL to hardware for every chassis that is a member of *groupID*. Before using this command, use the **set** commands for streams, capture and filter parameters on each port. The advantage of using this command is that the entire chassis chain configuration can be written into hardware at one time instead of writing into hardware for each stream, capture and filters on each port. Specific errors are:

- A port group with the specified groupID has not been created
- Network problem between the client and chassis

chassisChain enableStarTopology

This command is used if star topology is required. After a chassis chain exists and is valid, the chain topology can be specified.

By default the topology is daisy chain.

chassisChain disableStarTopology

Disables the star topology of a chassis chain that was previously enabled with enableStarTopology command.

chassisChain isValidForStarTopology

Validates if the collection of chassis can be chained in a star topology. Returns 1 if the start topology can be formed.

chassisChain isStarTopology

Verifies if the current chassis chain is configured as star topology. Returns 1 if the chassis is star.

EXAMPLES

See examples under *chassis*

SEE ALSO

chassis, *portGroup*

NAME - collisionBackoff

collisionBackoff - configure the collision backoff parameters for 10/100 ports

SYNOPSIS	collisionBackoff <i>sub-command options</i>
DESCRIPTION	The collisionBackoff command is used to configure the parameters for collision backoff operations for 10/100 ports.
STANDARD OPTIONS	
collisionConstant	Each successive retry operates by selecting a time slot over a range that doubles with each retry (2, 4, 8, ... 1024). This value controls the maximum number of time slots used. The values are powers of 2 from 0 through 1024. (<i>default = 10</i>)
continuousRetransmit <i>true / false</i>	If set, when a collision occurs, continuously retransmit the packet until the <i>maxRetryCount</i> is exhausted. (<i>default = false</i>)
maxRetryCount	The maximum number of retries for each packet. (<i>default = 16</i>)
random <i>true / false</i>	If set, when a collision occurs, wait a random amount of time before retrying the transmission. The <i>maxRetryCount</i> and <i>collisionConstant</i> values govern how often and long retries is attempted. (<i>default = true</i>)
COMMANDS	
	The collisionBackoff command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
collisionBackoff cget <i>option</i>	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the collisionBackoff command.
collisionBackoff config <i>option value</i>	Modify the Collision Backup configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available Collision Backoff options (see STANDARD OPTIONS) for port.
collisionBackoff get <i>chasID cardID portID</i>	Gets the current Collision Backoff configuration of the port with id <i>portID</i> on card <i>cardID</i> , chassis <i>chasID</i> . Call this command before calling collisionBackoff cget <i>option value</i> to get the value of the configuration option.
collisionBackoff setDefault	Sets to IxTclHal default values for all configuration options.
collisionBackoff set <i>chasID cardID portID</i>	Sets the Collision Backoff configuration of the port with id <i>portID</i> on card <i>cardID</i> , chassis <i>chasID</i> by reading the configuration option values set by the collisionBackoff config <i>option value</i> command.
EXAMPLES	See examples under forcedCollisions
SEE ALSO	forcedCollisions

NAME - conditionalStats

conditionalStats - works together with *conditionalTable* to configure and retrieve the flow detective stats from the port CPU.

SYNOPSIS	conditionalStats <i>sub-command options</i>
DESCRIPTION	The conditionalStats command is used to define the methods and parameters of the main configuration and stat retrieval object.
STANDARD OPTIONS	
fromPGID	First PGID in range to monitor.
toPGID	Last PGID in range to monitor.
fromStreamId	First stream ID in range to monitor.
toStreamId	Last stream ID in range to monitor.

COMMANDS	The conditionalStats command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	---

conditionalStats **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **conditionalStats** command.

conditionalStats **config** *option value*

Modify the Conditional Stats configuration options of the port. If no *option* is specified, returns a list describing all of the available Conditional Stats options (see STANDARD OPTIONS) for port.

conditionalStats **get** *chasID cardID portID conditionID*

Gets the current Conditional Stats for the port with *conditionID*, id *portID* on card *cardID*, chassis *chasID*.

Note: Add a delay (4000 ms) before the *conditionalStats get* sub-command.

conditionalStats **setDefault**

Sets to IxTclHal default values for all configuration options.

conditionalStats **set** *chasID cardID portID*

Sets the Conditional Stats configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **conditionalStats config option value** command.

conditionalStats **getrow** *RowIndex*

Returns list of the form {colName1 value} {colName2 value} etc, with the first colName1 pre-defined as the index column. Error returns empty string.

conditionalStats start *chassisID cardID portID conditionID*

Starts the collection of stats, based on the preset condition specified by *conditionalTable*.

Note: Add a delay (4000 ms) after *ixWriteConfigToHardware* and before the *conditionalStats start* sub-command.

conditionalStats stop *chassisID cardID portID conditionID*

Stops the collection of stats, based on the preset condition specified by *conditionalTable*.

EXAMPLES

```

package req IxTclHal

set hostname loopback

if {[ixConnectToChassis $hostname] == $::TCL_ERROR} {
    errorMsg "Error connecting to chassis"
    return 1
}

# skipping all the stream config stuff here...

conditionalTable setDefault
conditionalStats setDefault

set conditionId_max 32

conditionalTable config -columnNames {"Total Frames" "MaxLatency > 100000"}
conditionalTable config -sortingExpression "minLatency < 100000"
conditionalTable config -numResults100
if {[conditionalTable set $conditionId_max]} {
    errorMsg "Error setting conditionalTable - $::ixErrorInfo"
    return "FAIL"
}

set conditionId_min 42

conditionalTable config -columnNames {"Total Frames" "MinLatency < 1000"}
conditionalTable config -sortingExpression "minLatency < 1000"
conditionalTable config -numResults100
if {[conditionalTable set $conditionId_min]} {
    errorMsg "Error setting conditionalTable - $::ixErrorInfo"
    return "FAIL"
}

conditionalStats config -fromPGID 0
conditionalStats config -toPGID 10000
if {[conditionalStats set $chassis $card $port]} {
    errorMsg "Error configuring conditionalStats on port $chassis $card $port"
    return "FAIL"
}

ixClearStats portList
ixStartTransmit portList

```

```

if {[conditionalStats start $chassis $card $port
$conditionId_min]} {
    errorMsg "Error starting conditionalStats on port
$chassis $card $port"
    return "FAIL"
}

# maybe wait for a bit to let some stats accummulate..?
after 2000

# when you're ready, read some stats but just for the first 5
rows, let's say...
set fromRowIndex 0
set toRowIndex 5
if {[conditionalStats get $chassis $card $port $conditionId]} {
    errorMsg "Error getting conditionalStats on port
$chassis $card $port, condition = $conditionId"
    break
}
# row stats to be returned in the format:
#     {{rowIndex $rowNumber} {$colName1 $colValue1} {$colName2
$colValue2} ... {$colNameN $colValueN}}
foreach {set row 0} {$row < [conditionalTable cget -numResults]} {
{incr row} {
    set rowList [conditionalStats.getRow $row]
    if {[llength $rowList]} {
        errorMsg "Hmmm... no stats for $row..."
        break
    }
    array set rowArray [join $rowList]

    foreach {columnName value} [join $rowList] {
        ixPuts -nownewline [format "%-30s\t" $name]
    }
    ixPuts
    foreach columnName [array names $rowArray] {
        ixPuts -nonewline [format "%-30ld\t"
$rowArray($columnName)]
    }
    ixPuts
}
}

after $abit

# so now you decide to look at stats from a different set of
conditions...
# so I assume you have to stop & start different ones...?
if {[conditionalStats stop $chassis $card $port $conditionId_min]} {
    errorMsg "Error stopping conditionalStats on port
$chassis $card $port"
    return "FAIL"
}

if {[conditionalStats start $chassis $card $port
$conditionId_max]} {
    errorMsg "Error starting conditionalStats on port
$chassis $card $port"
    return "FAIL"
}

```

SEE ALSO

conditionalTable

NAME - conditionalTable

conditionalTable - works together with *conditionalStats* to configure and retrieve the flow detective stats from the port CPU.

SYNOPSIS	conditionalTable <i>sub-command options</i>									
DESCRIPTION	The conditionalTable command is used to configure and manipulate the table of conditional stats (flow detective stats).									
STANDARD OPTIONS										
columnNames	List of the names of the columns to retrieve.									
enableAggregation	Enables/disables aggregation mode. Default = disabled.									
filterExpression	The expression used for filtering the results. PGIDs for which the filter returns 0 is not included in the results.									
firstIndex	Only applies when in aggregation mode. The first bucket index to monitor. Default = 1.									
firstResult	Either the first PGID to be reported (after sorting and filtering) or, if in aggregated mode, the first bucket to report.									
lastIndex	Only applies when in aggregation mode. The last bucket index to monitor. A value of -1 means 'until the end' or 'all of them'.									
mask	PGID mask to mask the filter PGIDs down further. Default = no mask.									
numResults	Total number of results, or rows, to be reported (after sorting and filtering).									
sort	Sorting direction::									
	<table border="1"> <thead> <tr> <th>Option</th><th>Value</th><th>Usage</th></tr> </thead> <tbody> <tr> <td><i>conditionalTableSortDescending</i></td><td>0</td><td>(default) descending sort order</td></tr> <tr> <td><i>conditionalTableSortAscending</i></td><td>1</td><td>ascending sort order</td></tr> </tbody> </table>	Option	Value	Usage	<i>conditionalTableSortDescending</i>	0	(default) descending sort order	<i>conditionalTableSortAscending</i>	1	ascending sort order
Option	Value	Usage								
<i>conditionalTableSortDescending</i>	0	(default) descending sort order								
<i>conditionalTableSortAscending</i>	1	ascending sort order								
sortingExpression	The expression used for sorting the PGIDs.									
COMMANDS	The conditionalTable command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.									
conditionalTable cget option	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the conditionalTable command.									
conditionalTable config option value	Modify the Collision Backup configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available Conditional Table options (see STANDARD OPTIONS) for port.									

conditionalTable **get *requestID***

Gets the conditional data associated with the parameter *conditionID*.

conditionalTable **setDefault**

Sets to IxTclHal default values for all configuration options.

conditionalTable **set *requestID***

Sets the conditional data associated with the parameter *conditionID*.

conditionalTable **removeAll *requestID***

Removes all conditions.

EXAMPLES

See example under *conditionalStats*.

SEE ALSO

conditionalStats

NAME - customOrderedSet

customOrderedSet - configure a custom message for link fault signaling

SYNOPSIS customOrderedSet *sub-command options*

DESCRIPTION The **customOrderedSet** used to define the contents of two types of custom ordered sets: type A or type B. These messages are inserted into a transmitted stream with the *linkFaultSignaling* command.

STANDARD OPTIONS

blockType The block type of the message. (*default = 0x4B*)

byte1 - byte7 The remaining bytes of the message. (*default = all 0's, except byte3=1*)

syncBits The sync bits for the message. (*default = 2*)

COMMANDS The *customOrderedSet* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

customOrderedSet cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *customOrderedSet* command.

customOrderedSet config option value

Modify the configuration options of the ordered set type. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

customOrderedSet get orderedSetType

Gets the current configuration of the indicated set, which should be one of.

Option	Value	Usage
<i>linkFaultOrderedSetTypeA</i>	0	Set type A.
<i>linkFaultOrderedSetTypeB</i>	1	Set type B.

customOrderedSet set orderedSetType

Sets the current configuration of the indicated ordered set, one of.

Option	Value	Usage
<i>linkFaultOrderedSetTypeA</i>	0	Set type A.
<i>linkFaultOrderedSetTypeB</i>	1	Set type B.

customOrderedSet setDefault

Sets to IxTclHal default values for all configuration options. Note: Both the type A and type B sets are cleared.

EXAMPLES See examples under *linkFaultSignaling*.

SEE ALSO *linkFaultSignaling*

NAME - dataIntegrity

dataIntegrity - configure the Data Integrity parameters.

SYNOPSIS

dataIntegrity *sub-command options*

DESCRIPTION

The **dataIntegrity** command is used to configure the parameters for Data Integrity operations for Gigabit and OC-12/OC-48 ports. Data integrity values are additional checksums taken over a subset of a packet. In order for data integrity to operate, `receiveMode portRxDataIntegrity` must be performed (and committed).

Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in [atmHeader](#). The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command are with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

STANDARD OPTIONS

enableTimeStamp
true/false

For receive-mode only. Indicates that the received data integrity packets are expected to have a 48-bit timestamp before the FCS value. (*default = false*)

insertSignature
true/false

For transmit-mode only. Inserts the data integrity signature into the transmitted stream. (*default = false*)

signature

In the transmitted packet, the signature uniquely identifies the transmitted packet as one destined for receive port data integrity filtering .. On the receive port, the signature is used to filter only those packets that have a matching signature. (*default = '08 71 18 05'*)

signatureOffset

The offset, within the packet, of the data integrity signature. (*default = 40*)

floatingTimestampAnd
DataIntegrityMode

Enables adding timestamp as part of floating instrumentation header, and addresses similar issue in Data Integrity checking. (*default = dataIntegrityNumberOfBytesFromEndOfFrame*)

Option	Value	Usage
dataIntegrityNumberOfBytesFromEndOfFrame	0	(default) See numBytesFromEndOfFrame option, below
dataIntegrityPayloadLength	1	See payloadLength option, below

numBytesFromEndOf
Frame

Specify the number of padding bytes needed from the end of the frame. The number of padding bytes remains fixed with changing frame sizes. (*default = 4*)

payloadLength

Specify the fixed data integrity payload length. This length will not change with changing frame sizes. (*default = 0*)

COMMANDS

The **dataIntegrity** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

dataIntegrity cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **dataIntegrity** command.

dataIntegrity config *option value*

Modify the Data Integrity configuration options of the port. If no *option* is specified, returns a list describing all of the available Data Integrity options (see STANDARD OPTIONS) for port.

dataIntegrity get CircuitTx *chasID cardID portID [circuitID] streamID*

Gets the current configuration of the stream with id *streamID* in the circuit with *circuitID* on port *portID*, card *cardID*, chassis *chasID* from its hardware.

dataIntegrity get QueueTx *chasID cardID portID [queueID] streamID*

Gets the current configuration of the stream with id *streamID* in the queue with *queueID* on port *portID*, card *cardID*, chassis *chasID* from its hardware.

dataIntegrity get Rx *chasID cardID portID*

Gets the current receive Data Integrity configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Call this command before calling **dataIntegrity cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

dataIntegrity get Tx *chasID cardID portID streamID [streamType]*

Gets the current transmit Data Integrity configuration of the stream with id *portID* on card *cardID*, chassis *chasID*, stream *streamID*.

In the first form, the *queueID* indicates the particular queue for load modules which use multiple queues, such as ATM cards.

In the second form, the type of stream (stream or flow) is selected. One of.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) Both streams and flows. This option can be used for ports that do not use flows.
<i>streamSequenceTypeStreams</i>	1	Stream only.
<i>streamSequenceTypeFlows</i>	1	Flow only.

Call this command before calling **dataIntegrity cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- The stream does not exist

dataIntegrity setCircuitTx *chasID cardID portID [circuitID] streamID*

Sets the configuration of the stream with id *streamID* on its circuit *circuitID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL by reading the configuration option values set by the **dataIntegrity config option value** command.

dataIntegrity setDefault

Sets to IxTclHal default values for all configuration options.

dataIntegrity setQueueTx *chasID cardID portID [queueID] streamID*

Sets the configuration of the stream with id *streamID* on its queue *queueID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL by reading the configuration option values set by the **dataIntegrity config option value** command.

dataIntegrity setRx *chasID cardID portID*

Sets the receive Data Integrity configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **dataIntegrity config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

dataIntegrity setTx *chasID cardID portID streamID [streamType]*

Sets the transmit Data Integrity configuration of the stream with id *portID* on card *cardID*, chassis *chasID*, and stream *streamID* by reading the configuration option values set by the **dataIntegrity config option value** command.

In the first form, the *queueID* indicates the particular queue for load modules which use multiple queues, such as ATM cards.

In the second form, the type of stream (stream or flow) is selected. One of.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) Both streams and flows. This option can be used for ports that do not use flows.
<i>streamSequenceTypeStreams</i>	1	Stream only.
<i>streamSequenceTypeFlows</i>	1	Flow only.

After calling this command, the Data Integrity configuration should be committed to hardware using **stream write** or **ixWriteConfigToHardware** commands. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting
- The stream does not exist

EXAMPLES

```
package require IxTclHal
```

```

# In this example we'll use an OC12c card with port 1 (transmit)
is
# directly connected to port 2 (receive)

# Data integrity is transmitted with a time stamp and received and
checked
# by the receive port

set host galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assumes that card 2 is a OC12c card with ports 1 and 2 directly
connected
set card 2
set txPort 1
set rxPort 2

# Useful port lists
set portList [list [list $chas $card $txPort] \
                    [list $chas $card $rxPort]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Set up Transmit Port

# Nothing special about the port
port setFactoryDefaults $chas $card $txPort
port setDefault
port set $chas $card $txPort

# One port must use recovered clock and the other not
sonet setDefault
sonet config -useRecoveredClock true
sonet set $chas $card $txPort

# Stream: 100,000 packets

```

```

stream setDefault
stream config -numFrames          100000
stream config -framesize          4148
stream config -fir                true
stream config -dma                stopStream
stream config -percentPacketRate 100
stream config -rateMode           usePercentRate
stream set $chassis $card $txPort 1

dataIntegrity setDefault
dataIntegrity config -insertSignature true
dataIntegrity setTx $chassis $card $txPort 1

# Set up the Receive Port

# Set the receive mode to data integrity
port setFactoryDefaults $chassis $card $rxPort
port setDefault
port config -receiveMode portRxDataIntegrity
port set $chassis $card $rxPort

# This port does not use recovered clock
sonet setDefault
sonet config -useRecoveredClock false
sonet set $chassis $card $rxPort

# Enable receive mode DI and expect a time stamp
dataIntegrity setDefault
dataIntegrity config -enableTimeStamp true
dataIntegrity setRx $chassis $card $rxPort

# Commit to hardware
ixWritePortsToHardware portList

# Make sure link is up
after 1000
ixCheckLinkState portList

# Clear stats on receive side and start transmitting
ixClearPortStats $chassis $card $rxPort
ixStartPortTransmit $chassis $card $txPort
after 1000

# Wait until done
ixCheckPortTransmitDone $chassis $card $txPort

# Get the DI frames received and errors
stat get allStats $chassis $card $rxPort
set diFrames [stat cget -dataIntegrityFrames]
set diErrors [stat cget -dataIntegrityErrors]

ixPuts "$diFrames Data Integrity Frames received, $diErrors
errors"

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

NAME - **dcc**

dcc - configure the dcc parameters on a port.

SYNOPSIS

dcc *sub-command options*

DESCRIPTION

The **dcc** command is used to configure the DCC (Data Communications Channel) parameters associated with a SONET port. The selection of where the DCC bytes are written (SOH or LOH), the type of CRC and the time fill byte to be used are controlled.

STANDARD OPTIONS

crc

Selects the type of CRC generated in the DCC data. Available option values are:

Option	Value	Usage
<i>dccCrc16</i>	0	(default) 16-bit CRC
<i>dccCrc32</i>	1	32-bit CRC

overheadBytes

Selects the placement of DCC bytes in the SONET overhead. Available option values are:

Option	Value	Usage
<i>dccSoh</i>	0	(default) Data is placed in the section overhead.
<i>dccLoh</i>	1	Data is placed in the line overhead.

timeFill

Selects the type of fill byte to use. Available option values are:

Option	Value	Usage
<i>dccTimeFillFlag7E</i>	0	(default) Fill blank time with 0x7E bytes.
<i>dccTimeFillMarkIdle</i>	1	Fill blank time with 0xFF bytes.

COMMANDS

The **dcc** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

dcc cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **dcc** command.

dcc config *option value*

Modify the configuration options of the **dcc**. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **dcc**.

dcc get *chasID cardID portID*

Gets the current configuration of the **dcc** for port with id *portID* on card *cardID*, chassis *chasID*. from its hardware. Call this command before calling **dcc cget *option value*** to get the value of the configuration option. In order for this

command to succeed, the port must either be unowned, or you must be logged in as the owner of the port. Specific errors are:

- No connection to a chassis
- Invalid port number

dcc set *chasID cardID portID*

Sets the configuration of the dcc in IxHAL for port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **dcc config *option value*** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

dcc setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 27
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Need to be in a DCC mode via port
port setFactoryDefaults $chas $card $port
port config -transmitMode portTxModeDccStreams
if [port set $chas $card $port] {
```

```
        ixPuts "Could not port set $chas $card $port"
    }

# Set to 32-bit CRC and use of Line Overhead
dcc setDefault
dcc config -crc                      dccCrc32
dcc config -overheadBytes      dccLoh
if [dcc set    $chas $card $port] {
    ixPuts "Could not dcc set $chas $card $port"
}

ixWriteConfigToHardware portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

port, stream

NAME - dhcp

dhcp - configure the DHCP parameters on a stream of a port.

SYNOPSIS	<code>dhcp sub-command options</code>
-----------------	---------------------------------------

DESCRIPTION	The dhcp command is used to configure the DHCP parameters. Refer to RFC 2131 and RFC 2132 for detailed descriptions of DHCP. Note that <i>stream</i> get must be called before this command's get sub-command.
--------------------	---

STANDARD OPTIONS

bootFileName	Boot file name, null terminated string; "generic" name or null in DHCPDISCOVER, fully qualified folder-path name in DHCPOFFER.									
clientHwAddr	Client hardware address. (<i>default</i> = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00)									
clientIpAddr	Client IP address. Only filled in if client is in BOUND, RENEW or REBINDING state and can respond to ARP requests. (<i>default</i> = 0.0.0.0)									
flags	Available option values are:									
	<table border="1"> <thead> <tr> <th>Option</th> <th>Value</th> <th>Usage</th> </tr> </thead> <tbody> <tr> <td><code>dhcpNoBroadcast</code></td> <td>0</td> <td>(<i>default</i>) Do not broadcast</td> </tr> <tr> <td><code>dhcpBroadcast</code></td> <td>0x8000</td> <td>Broadcast</td> </tr> </tbody> </table>	Option	Value	Usage	<code>dhcpNoBroadcast</code>	0	(<i>default</i>) Do not broadcast	<code>dhcpBroadcast</code>	0x8000	Broadcast
Option	Value	Usage								
<code>dhcpNoBroadcast</code>	0	(<i>default</i>) Do not broadcast								
<code>dhcpBroadcast</code>	0x8000	Broadcast								
hops	Set to zero by client. (<i>default</i> = 0)									
hwLen	Hardware address length. (<i>default</i> = 6)									
hwType	Hardware address types. Available option values are:									

Option	Value	Usage
<code>dhcpEthernet10Mb</code>	1	(<i>default</i>) Ethernet 10 Mb
<code>dhcpEthernet3Mb</code>	2	Ethernet 3 Mb
<code>dhcpAmateur</code>	3	Amateur radio AX.25
<code>dhcpProteon</code>	4	Proteon ProNET token ring
<code>dhcpChaos</code>	5	Chaos
<code>dhcpIEEE</code>	6	IEEE 802 networks
<code>dhcpARCNET</code>	7	ARCNET
<code>dhcpHyperchannel</code>	8	Hyperchannel
<code>dhcpLanstar</code>	9	LanStar
<code>dhcpAutonet</code>	10	Autonet short address
<code>dhcpLocalTalk</code>	11	LocalTalk
<code>dhcpLocalNet</code>	12	LocalNet
<code>dhcpUltraLink</code>	13	Ethernet
<code>dhcpSMDS</code>	14	SMDS

Option	Value	Usage
<i>dhcpFrameRelay</i>	15	Frame Relay
<i>dhcpATM1</i>	16	ATM
<i>dhcpHDLC</i>	17	HDLC
<i>dhcpFibreChannel</i>	18	Fibre Channel
<i>dhcpATM2</i>	19	ATM
<i>dhcpSerialLine</i>	20	Serial Line
<i>dhcpATM3</i>	21	ATM

opCode

Operation code. Available option values are:

Option	Value	Usage
<i>dhcpBootRequest</i>	1	(default) BOOTP request
<i>dhcpBootReply</i>	2	BOOTP reply

optionCode

The *code* field of the options section of the DHCP frame. Available codes are:

Option	Value	Usage
<i>dhcpPad</i>	0	(default)
<i>dhcpEnd</i>	255	
<i>dhcpSubnetMask</i>	1	
<i>dhcpTimeOffset</i>	2	
<i>dhcpGateways</i>	3	
<i>dhcpTimeServer</i>	4	
<i>dhcpNameServer</i>	5	
<i>dhcpDomainNameServer</i>	6	
<i>dhcpLogServer</i>	7	
<i>dhcpCookieServer</i>	8	
<i>dhcpLPRServer</i>	9	
<i>dhcpImpressServer</i>	10	
<i>dhcpResourceLocationServer</i>	11	
<i>dhcpHostName</i>	12	
<i>dhcpBootFileSize</i>	13	
<i>dhcpMeritDumpFile</i>	14	
<i>dhcpDomainName</i>	15	
<i>dhcpSwapServer</i>	16	
<i>dhcpRootPath</i>	17	
<i>dhcpExtensionPath</i>	18	

IP Layer Parameters per Host

Option	Value	Usage
<i>dhcpIpForwardingEnable</i>	19	
<i>dhcpNonLocalSrcRoutingEnable</i>	20	
<i>dhcpPolicyFilter</i>	21	
<i>dhcpMaxDatagramReassemblySize</i>	22	

Option	Value	Usage
dhcpDefaultIpTTL	23	
dhcpPathMTUAgingTimeout	24	

IP Layer Parameters per Interface

Option	Value	Usage
dhcpPathMTUPlateauTable	25	
dhcpInterfaceMTU	26	
dhcpAllSubnetsAreLocal	27	
dhcpBroadcastAddress	28	
dhcpPerformMaskDiscovery	29	
dhcpMaskSupplier	30	
dhcpPerformRouterDiscovery	31	
dhcpRouterSolicitAddr	32	
dhcpStaticRoute	33	

Link Layer Parameters per Interface

Option	Value	Usage
dhcpTrailerEncapsulation	34	
dhcpARPCacheTimeout	35	
dhcpEthernetEncapsulation	36	

TCP Parameters

Option	Value	Usage
dhcpTCPDefaultTTL	37	
dhcpTCPKeepAliveInterval	38	
dhcpTCPKeepGarbage	39	

Application And Service Parameters

Option	Value	Usage
dhcpNISDomain	40	
dhcpNISServer	41	
dhcpNTPServer	42	
dhcpVendorSpecificInfo	43	
dhcpNetBIOSNameSrv	44	
dhcpNetBIOSDatagramDistSrv	45	
dhcpNetBIOSNodeType	46	
dhcpNetBIOSScope	47	
dhcpXWinSysFontSrv	48	

DHCP Extensions

Option	Value	Usage
dhcpRequestedIPAddr	50	

Option	Value	Usage
dhcpIPAddrLeaseTime	51	
dhcpOptionOverload	52	
dhcpTFTPSvrName	66	
dhcpBootFileName	67	
dhcpMessageType	53	
dhcpSrvIdentifier	54	
dhcpParamRequestList	55	
dhcpMessage	56	
dhcpMaxMessageSize	57	
dhcpRenewalTimeValue	58	
dhcpRetryCountValue	4	The configurable retry count of the DHCP Extention server.
dhcpRebindingTimeValue	59	
dhcpVendorClassId	60	
dhcpClientId	61	
dhcpXWinSysDisplayMgr	49	
dhcpNISplusDomain	64	
dhcpNISplusServer	65	
dhcpMobileIPHomeAgent	68	
dhcpSMTPSvr	69	
dhcpPOP3Svr	70	
dhcpNNTPSvr	71	
dhcpWWWSvr	72	
dhcpDefaultFingerSvr	73	
dhcpDefaultIRCSvr	74	
dhcpStreetTalkSvr	75	
dhcpSTDASvr	76	
dhcpAgentInformationOption	82	
dhcpNetwareIpDomain	62	
dhcpNetworkIpOption	63	

optionData

The data in the options section of the DHCP frame. Option data may either be set as a single value (for example, 255.255.255.0), a stream of bytes (for example, {01 03 06 0F 2C 2E 2F 39}) or as a list of enumerated values (for example, [list dhcpSubnetMask dhcpGateways dhcpDomainNameServer]) (default = {})

optionDataLength

The length of the data in the options section of the DHCP frame. (default = 0)

relayAgentIpAddr

Relay agent IP address, used in booting by a relay agent. (default = 0.0.0.0)

seconds

Seconds elapsed since client began address acquisition or renewal process. (default = 0)

serverHostName

Optional server host name, null terminated string. (default = "")

serverIpAddr

IP address of next server to use in bootstrap; returned in DHCPOFFER, DHCPACK by server. (default = 0.0.0.0)

transactionID Random number chosen by client and used by the client and server to associate messages and responses between a client and a server. (*default = 0*)

yourIpAddr 'your' (client) IP address. (*default = 0.0.0.0*)

COMMANDS The **dhcp** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

dhcp cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **dhcp** command.

dhcp config option value

Modify the configuration options of the dhcp. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for arp.

dhcp decode capFrame [chasID cardID portID]

Decodes a captured frame in the capture buffer and updates TclHal. **dhcp getOption** command can be used after decoding to get the option data. Specific errors are:

- No connection to a chassis
- The captured frame is not a valid DHCP packet

dhcp get chasID cardID portID

Gets the current configuration of the dhcp frame for port with id *portID* on card *cardID*, chassis *chasID*. from its hardware. Note that *stream* get must be called before this command's get sub-command. Call this command before calling **dhcp cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

dhcp getFirstOption

The first option is retrieved. Specific errors are:

- There are no more entries in the list.

dhcp getNextOption

The next option is retrieved. Specific errors are:

- *getFirstOption* has not been called yet.
- There are no more entries in the list.

dhcp getOption optionCodeType

Gets the option data for *optionCodeType*. Specific errors are:

- There is no option data for the *optionCodeType*.

dhcp set chasID cardID portID

Sets the configuration of the dhcp in IxHAL for port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **dhcp config option value** command. Specific errors are:

- No connection to a chassis

- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

dhcp setDefault

Sets to IxTclHal default values for all configuration options.

dhcp setOption *optionCodeType*

Sets the option data for *optionCodeType*. Specific errors are:

- The configured parameters are not valid for this port

EXAMPLES

```
package require IxTclHal

# In this example we'll generate a DHCP response packet
# with a number of option fields

set host galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assume card to be used is in slot 1
set card 1
set port 1
set portList [list [list $chas $card $port]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Put the port in loopback mode
port setFactoryDefaults $chas $card $port
port setDefault

# Stream: 1 packet at 1%, with framesize large enough to hold all
options
stream setDefault
```

```

stream config -numFrames          1
stream config -dma                stopStream
stream config -rateMode           usePercentRate
stream config -percentPacketRate  1
stream config -framesize          512

# Set up IP: udp with 494 byte packet
ip setDefault
ip config -ipProtocol             udp
ip config -totalLength            494
ip set $chas $card $port

# Set up protocol
protocol setDefault
protocol config -name              ipV4
protocol config -appName            Dhcp

# Set up UDP
udp setDefault
udp config -sourcePort            bootpClientPort
udp config -destPort              bootpServerPort
udp set $chas $card $port

# Setup DHCP with options
dhcp setDefault
dhcp config -opCode               dhcpBootReply
dhcp config -hwType               dhcpEthernet10Mb
dhcp config -hwLen                6
dhcp config -flags                dhcpBroadcast
dhcp config -yourIpAddr           192.168.18.154
dhcp config -serverIpAddr          192.168.18.2
dhcp config -clientHwAddr          {01 02 03 04 05 06}

# Options
dhcp config -optionData           255.255.255.0
dhcp setOption dhcpSubnetMask
dhcp config -optionData           192.168.18.254
dhcp setOption dhcpRouter
dhcp setOption dhcpGateways
dhcp config -optionData           192.168.18.2
dhcp setOption dhcpNameServer
dhcp config -optionData           widgets.com
dhcp setOption dhcpDomainName
dhcp config -optionData           {cc ee 22 11 33 ff}
dhcp setOption dhcpNetBIOSScope
dhcp config -optionData [list dhcpSubnetMask \
                        dhcpGateways \
                        dhcpDomainNameServer \
                        dhcpDomainName \
                        dhcpNetBIOSNameSrv \
                        dhcpNetBIOSNodeType \
                        dhcpNetBIOSScope]
dhcp setOption dhcpParamRequestList

dhcp set $chas $card $port

stream set $chas $card $port 1
port set $chas $card $port

ixWritePortsToHardware portList
# Let go of the ports that we reserved
ixClearOwnership $portList

```

```
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

capture, *captureBuffer*

NAME - dhcpV4DiscoveredInfo

dhcpV4DiscoveredInfo - view discovered DHCP information

SYNOPSIS	dhcpV4DiscoveredInfo <i>sub-command options</i>
-----------------	---

DESCRIPTION	The dhcpV4DiscoveredInfo command is used retrieve the DHCP negotiated values. The IP address, gateway address, prefix length and renewal timer are all visible in <i>STANDARD OPTIONS</i> ; all other options are available as TLVs obtained by using <i>getFirstTlv</i> , <i>getNextTlv</i> and <i>getTlv</i> .
--------------------	---

STANDARD OPTIONS

gatewayIpAddress	<i>Read-only</i> . The gateway address from the DHCP server.
ipAddress	<i>Read-only</i> . The IP address from the DHCP server.
prefixLength	<i>Read-only</i> . The prefix/mask length for the network, from the DHCP server.
leaseDuration	<i>Read-only</i> . The lease timer set by the DHCP server.

COMMANDS	The <i>dhcpV4DiscoveredInfo</i> command is invoked with the following sub-commands. If no sub-command is used, returns a list of all sub-commands available.
-----------------	--

dhcpV4DiscoveredInfo cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *dhcpV4Tlv* command.

dhcpV4DiscoveredInfo getFirstTlv

The first TLV is retrieved. The values are available in the *dhcpV4Tlv* command. Specific errors are:

- There are no entries in the list.

dhcpV4DiscoveredInfo getNextTlv

The next TLV is retrieved. The values are available in the *dhcpV4Tlv* command. Specific errors are:

- There are no more entries in the list.

dhcpV4DiscoveredInfo getTlv index

The TLV at the specified *index* is retrieved. The *index* of the first entry is 1. The values are available in the *dhcpV4Tlv* command. Specific errors are:

- The *index*'d entry does not exist in the list.
- Invalid index.
- There are no entries in the list.

EXAMPLES	See example under <i>interfaceTable</i> .
-----------------	---

SEE ALSO

interfaceTable, *interfaceEntry*, *dhcpV4Properties*, *dhcpV4Tlv*

NAME - dhcpV4Properties

dhcpV4Properties - describe/view DHCP properties for an interface entry

SYNOPSIS

dhcpV4Properties *sub-command options*

DESCRIPTION

The **dhcpV4Properties** command is used in two contexts:

- When a new *interfaceEntry* is added to the *interfaceTable*, the values from this command are associated with the entry.
- When an existing interface is retrieved with *interfaceTable get*Interface* and the *enableDhcp* option in the *interfaceEntry* is true. The values associated with the interface entry are made available in this command.

Four standard DHCP options are set in the *STANDARD OPTIONS* below, others may be set as TLVs using *dhcpV4Tlv* and the *addTlv* sub-command.

STANDARD OPTIONS

clientId

The client identifier, which must be unique for the subnet that the interface is connected to. If this is not set, the MAC address of the protocol interface entry is used. (*default = “”*)

renewTimer

The requested value for the renewal time, in seconds. The actual value used is the lower of this value and the release time set by the DHCP server. (*default = 0*)

relayAgentAddress

The IP address of the DHCPv4 relay agent. This is only valid for unconnected interfaces.

relayDestination Address

The destination IP address for DHCPv4 relay messages. This is only valid for unconnected interfaces.

retryCount

The configurable retry count of the DHCP server. (*default = 4*)

serverId

If specified as a non-zero value, DHCP negotiation only occurs with a particular server. This entry should be specified as an IPv4 address. (*default = 0.0.0.0*)

vendorId

The vendor Id associated with the client. (*default = “”*)

COMMANDS

The *dhcpV4Properties* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

dhcpV4Properties addTlv

The DHCP TLV specified in *dhcpV4Tlv* is added to this property set. Specific errors are:

- Invalid TLV parameters.

dhcpV4Properties cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *dhcpV4Properties* command.

dhcpV4Properties config *option value*

Modify the configuration options of the *dhcpV4Properties*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *dhcpV4Properties*.

dhcpV4Properties delTlv *index*

The TLV associated with this DHCP property set at the specified *index* is deleted. The *index* of the first entry is 1. The values are available in the *dhcpV4Tlv* command. Specific errors are:

- The *index*'d entry does not exist in the list.
- Invalid index.

dhcpV4Properties getFirstTlv

The first TLV associated with this DHCP property set is retrieved. The values are available in the *dhcpV4Tlv* command. Specific errors are:

- There are no entries in the list.

dhcpV4Properties getNextTlv

The next TLV associated with this DHCP property set is retrieved. The values are available in the *dhcpV4Tlv* command. Specific errors are:

- There are no more entries in the list.

dhcpV4Properties getTlv *index*

The TLV associated with this DHCP property set at the specified *index* is retrieved. The *index* of the first entry is 1. The values are available in the *dhcpV4Tlv* command. Specific errors are:

- The *index*'d entry does not exist in the list.

dhcpV4Properties removeAllTlvs

Deletes all of the TLVs associated with this DHCP property set.

dhcpV4Properties setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *interfaceTable*.

SEE ALSO

interfaceTable, *interfaceEntry*, *dhcpV4DiscoveredInfo*, *dhcpV4Tlv*

NAME - dhcpV4Tlv

dhcpV4Tlv - describe/view a single DHCP option

SYNOPSIS

dhcpV4Tlv *sub-command options*

DESCRIPTION

The **dhcpV4Tlv** command is used in three contexts:

- When a new TLV (type-length-value) is added to a *dhcpV4Properties* set. Values are taken from the options in this command.
- When an existing TLV is retrieved with *dhcpV4Properties get*Tlv*. The TLV values are visible in this command.
- When the negotiated DHCP options are retrieved with *interfaceTable getDhcpV4DiscoveredInfo* and the *dhcpV4DiscoveredInfo* command. The TLV values are visible in this command.

A TLV should include DHCP options defined in RFC 2132.

STANDARD OPTIONS

type

The type of the DHCP option. One of the values defined in RFC 2132. (*default = 0*)

value

A string consisting of hexadecimal characters. Each pair of characters defines a byte value. The length of the TLV is set from the length of the value string, divided by 2. (*default = “”*)

COMMANDS

The *dhcpV4Tlv* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

dhcpV4Tlv cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *dhcpV4Tlv* command.

dhcpV4Tlv config option value

Modify the configuration options of the *dhcpV4Tlv*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *dhcpV4Tlv*.

dhcpV4Tlv setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *interfaceTable*.

SEE ALSO

interfaceTable, *interfaceEntry*, *dhcpV4DiscoveredInfo*, *dhcpV4Properties*

NAME - **dhcpV6DiscoveredInfo**

dhcpV6DiscoveredInfo - view discovered DHCPv6 information

SYNOPSIS

dhcpV6DiscoveredInfo *sub-command options*

DESCRIPTION

The **dhcpV6DiscoveredInfo** command is used retrieve the DHCPv6 negotiated values. Options are available as TLVs obtained by using *getFirstTlv*, *getNextTlv* and *getTlv*.

STANDARD OPTIONS

discoveredAddressList	<i>Read-only</i> . A list of discovered IP addresses.
iaRebindTime	<i>Read-only</i> . The rebind timer value specified by the DHCPv6 Server, in seconds.
iaRenewTime	<i>Read-only</i> . The renew timer value specified by the DHCPv6 Server, in seconds.

COMMANDS

The *dhcpV4DiscoveredInfo* command is invoked with the following sub-commands. If no sub-command is used, returns a list of all sub-commands available.

dhcpV6DiscoveredInfo cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *dhcpV4DiscoveredInfo* command.

dhcpV6DiscoveredInfo getFirstTlv

The first TLV is retrieved. The values are available in the *dhcpV6Tlv* command. Specific errors are:

- There are no entries in the list.

dhcpV6DiscoveredInfo getNextTlv

The next TLV is retrieved. The values are available in the *dhcpV6Tlv* command. Specific errors are:

- There are no more entries in the list.

dhcpV6DiscoveredInfo getTlv index

The TLV at the specified *index* is retrieved. The *index* of the first entry is 1. The values are available in the *dhcpV6Tlv* command. Specific errors are:

- The *index*'d entry does not exist in the list.
- Invalid index.
- There are no entries in the list.

dhcpV6DiscoveredInfo setDefault index

Sets the DHCPv6 values back to their defaults.

EXAMPLES

See example under *interfaceTable*.

SEE ALSO

interfaceTable, *interfaceEntry*, *dhcpV6Properties*, *dhcpV6Tlv*

NAME - dhcpV6Properties

dhcpV6Properties - describe/view DHCP properties for an interface entry

SYNOPSIS

dhcpV6Properties *sub-command options*

DESCRIPTION

The **dhcpV6Properties** command is used in two contexts:

- When a new *interfaceEntry* is added to the *interfaceTable*, the values from this command are associated with the entry.
- When an existing interface is retrieved with *interfaceTable get*Interface* and the *enableDhcp* option in the *interfaceEntry* is true. The values associated with the interface entry are made available in this command.

Standard DHCPv6 options are set in the *STANDARD OPTIONS* below, others may be set as TLVs using *dhcpV6Tlv* and the *addTlv* sub-command.

STANDARD OPTIONS

iaID

The client identifier, which must be unique for the subnet that the interface is connected to. If this is not set, the MAC address of the protocol interface entry is used. (*default* = "")

iaType

The type of DHCPv6 address. Values are:

Option	Value	Usage
<i>dhcpV6IaTypeTemporary</i>	0	A temporary IA address.
<i>dhcpV6IaTypePermanent</i>	1	A permanent IA address.
<i>dhcpV6IaTypePrefixDelegation</i>	2	An address that carries a DHCPv6 prefix

relayLinkAddress

The IP address of the DHCPv6 relay link.

relayDestination Address

The IP address for DHCPv6 relay messages.

renewTimer

The requested value for the renewal time, in seconds. The actual value used is the lower of this value and the release time set by the DHCPv6 server. (*default* = 0)

COMMANDS

The *dhcpV6Properties* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

dhcpV6Properties addTlv

The DHCPv6 TLV specified in *dhcpV6Tlv* is added to this property set. Specific errors are:

- Invalid TLV parameters.

dhcpV6Properties cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *dhcpV6Properties* command.

dhcpV6Properties config option value

Modify the configuration options of the *dhcpV6Properties*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *dhcpV4Properties*.

dhcpV6Properties delTlv index

The TLV associated with this DHCPv6 property set at the specified *index* is deleted. The *index* of the first entry is 1. The values are available in the *dhcpV6Tlv* command. Specific errors are:

- The *index*'d entry does not exist in the list.
- Invalid index.

dhcpV6Properties getFirstTlv

The first TLV associated with this DHCPv6 property set is retrieved. The values are available in the *dhcpV6Tlv* command. Specific errors are:

- There are no entries in the list.

dhcpV6Properties getNextTlv

The next TLV associated with this DHCPv6 property set is retrieved. The values are available in the *dhcpV6Tlv* command. Specific errors are:

- There are no more entries in the list.

dhcpV6Properties getTlv index

The TLV associated with this DHCPv6 property set at the specified *index* is retrieved. The *index* of the first entry is 1. The values are available in the *dhcpV6Tlv* command. Specific errors are:

- The *index*'d entry does not exist in the list.

dhcpV6Properties removeAllTlvs

Deletes all of the TLVs associated with this DHCPv6 property set.

dhcpV6Properties setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *interfaceTable*.

SEE ALSO

interfaceTable, *interfaceEntry*, *dhcpV6DiscoveredInfo*, *dhcpV6Tlv*

NAME - *dhcpV6Tlv*

dhcpV6Tlv - describe/view a single DHCPv6 option

SYNOPSIS	<i>dhcpV6Tlv sub-command options</i>
-----------------	--------------------------------------

DESCRIPTION	The dhcpV6Tlv command is used in three contexts: <ul style="list-style-type: none"> • When a new TLV (type-length-value) is added to a <i>dhcpV6Properties</i> set. Values are taken from the options in this command. • When an existing TLV is retrieved with <i>dhcpV6Properties get*Tlv</i>. The TLV values are visible in this command. • When the negotiated DHCP options are retrieved with <i>interfaceTable getDhcpV4DiscoveredInfo</i> and the <i>dhcpV6DiscoveredInfo</i> command. The TLV values are visible in this command.
--------------------	---

A TLV should include DHCPv6 options defined in RFC 2132.

STANDARD OPTIONS

type	The type of the DHCPv6 option. One of the values defined in RFC 2132. (<i>default = 0</i>)
value	A string consisting of hexadecimal characters. Each pair of characters defines a byte value. The length of the TLV is set from the length of the value string, divided by 2. (<i>default = “”</i>)

COMMANDS	The <i>dhcpV6Tlv</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

dhcpV6Tlv cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *dhcpV6Tlv* command.

dhcpV6Tlv config option value

Modify the configuration options of the *dhcpV6Tlv*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *dhcpV6Tlv*.

dhcpV6Tlv setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES	See example under <i>interfaceTable</i> .
-----------------	---

SEE ALSO	<i>interfaceTable, interfaceEntry, dhcpV6DiscoveredInfo, dhcpV6Properties</i>
-----------------	---

NAME - discoveredAddress

discoveredAddress - access discovered IP addresses.

SYNOPSIS discoveredAddress *sub-command options*

DESCRIPTION The *discoveredAddress* command holds an IPv4 or IPv6 address associated with an interface (as retrieved in *discoveredList*) or the IPv4/IPv6 address associated with a neighbor (as retrieved in *discoveredNeighbor*).

STANDARD OPTIONS

ipAddress (*Read-only*) The retrieved IPv4 or IPv6 address, as a character string.

COMMANDS The *discoveredList* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

discoveredAddress cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *discoveredList* command.

EXAMPLES See examples under *interfaceTable*.

SEE ALSO

NAME - discoveredList

discoveredList - access discovered neighbors and interface addresses.

SYNOPSIS	discoveredList <i>sub-command options</i>
DESCRIPTION	<p>The <i>discoveredList</i> command must be preceded with use of three commands in the <i>interfaceTable</i> command: <i>sendRouterSolicitation</i>, <i>requestDiscoveredTable</i> and <i>getDiscoveredList</i>. The <i>discoveredList</i> command is used to look through two lists associated with an interface:</p> <ul style="list-style-type: none"> • Neighbor list: contains a list of discovered neighbors, each of which contains a MAC address and a list of IP addresses. • Address list: contains the list of IP addresses associated with the interface.
<hr/>	
STANDARD OPTIONS	
none	
<hr/>	
COMMANDS	<p>The discoveredList command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.</p>
discoveredList getFirstAddress	<p>Gets the first address associated with the interface, which can be accessed with the <i>discoveredAddress</i> command. Specific errors are:</p> <ul style="list-style-type: none"> • Required commands have not been called. • The list is empty.
discoveredList getFirstNeighbor	<p>Gets the data concerning the first discovered neighbor in the list, which can be accessed with the <i>discoveredNeighbor</i> command. Specific errors are:</p> <ul style="list-style-type: none"> • Required commands have not been called. • The list is empty.
discoveredList getNextAddress	<p>Gets the next address associated with the interface, which can be accessed with the <i>discoveredAddress</i> command. Specific errors are:</p> <ul style="list-style-type: none"> • <i>getFirstAddress</i> has not been called. • There are no more objects in the list.
discoveredList getNextNeighbor	<p>Gets the data concerning the next discovered neighbor in the list, which can be accessed with the <i>discoveredNeighbor</i> command. Specific errors are:</p> <ul style="list-style-type: none"> • <i>getFirstNeighbor</i> has not been called.

discoveredList **getNeighbor *ipAddress***

Gets the data concerning the discovered neighbor in the list which has an interface address that matches *ipAddress*. The neighbor can be accessed with the command. Specific errors are:

- There is no object with this ID.
- Required commands have not been called.

EXAMPLES

See examples under *interfaceTable*.

SEE ALSO

NAME - *discoveredNeighbor*

discoveredNeighbor - access discovered neighbors.

SYNOPSIS

discoveredNeighbor sub-command options

DESCRIPTION

The *discoveredNeighbor* command holds an entry for each neighbor discovered as a result of router discovery or neighbor discovery announcements. Each neighbor entry has:

- MAC address: the MAC address of the discovered interface.
- Router flag: if the neighbor is a router.
- Address list: a list of IP addresses associated with the neighbor's interface, accessed with the *discoveredAddress* command.

STANDARD OPTIONS

isRouter

(*Read-only*). Set to *true* if the neighbor is a router and *false* otherwise.

macAddress

(*Read-only*). The retrieved MAC address, as a character string in the form *XX:XX:XX:XX:XX:XX*.

COMMANDS

The *discoveredNeighbor* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

discoveredNeighbor cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *discoveredNeighbor* command.

discoveredNeighbor getFirstAddress

Gets the first address associated with the neighbor, which can be accessed with the *discoveredAddress* command. Specific errors are:

- Required commands have not been called.
- The list is empty.

discoveredNeighbor getNextAddress

Gets the next address associated with the neighbor, which can be accessed with the *discoveredAddress* command. Specific errors are:

- *getFirstAddress* has not been called.
- There are no more objects in the list.

EXAMPLES

See examples under *interfaceTable*.

SEE ALSO

discoveredList, *discoveredAddress*

NAME - encHeader

encHeader-inserts Encapsulation Extended Header (ENC Header) in a fibre channel packet

SYNOPSIS

encHeader *sub-command options*

DESCRIPTION

The Encapsulation Extended Header (ENC Header) is used to transmit frames between Inter-Fabric Routers when connected through an FC-SW-3 or FC-SW-4 compliant fabric. To preserve backward compatibility, the Inter-Fabric Routers appear as N_Ports to the FC-SW-3 or FC-SW-4 compliant Fabric.

STANDARD OPTIONS

destinationId

The Destination ID (D_ID) is a three-byte field (Word 0, Bits 23-0) that contains the address identifier of the destination Nx_Port.

routingControl

The R_CTL field is a one-byte field that contains routing bits and information bits to categorize the frame function.

This field is set to the value 52h to identify the IFR_Header.

sourceId

The Source ID (S_ID) is a three-byte field that contains the address identifier of the source Nx_Port.

csControlOrPriority

CS_CTL field is controlled by the CS_CTL/Priority Enable bit (F_CTL, bit 17).

frameControl

The Frame Control (F_CTL) field (Word 2, Bits 23-0) is a three-byte field that contains control information relating to the frame content. If an error in bit usage is detected, a reject frame (P_RJT) is transmitted in response with an appropriate reason code for Class 1, Class 2, and Class 6.

type

The data structure type is a one-byte field that identifies the protocol of the frame content for Data frames.

sequenceCount

The Sequence Count is a two-byte field that indicates the sequential order of Data frame transmission within a single Sequence or multiple consecutive Sequences for the same Exchange. The SEQ_CNT of the first Data frame of the first Sequence of the Exchange transmitted by either the Originator or Responder is binary zero. The SEQ_CNT of each subsequent Data frame in the Sequence is incremented by one.

dataFieldControl

Data Field Control (DF_CTL) is a one-byte field that specifies the presence of optional headers at the beginning of the Data_Field.

sequenceId

The Sequence ID (SEQ_ID) is a one-byte field (Word 3, Bits 31-24) assigned by the Sequence Initiator.

responderExchangeId

The Responder Exchange_ID is a two byte field assigned by the Responder that provides a unique, locally meaningful identifier at the Responder for an Exchange established by an Originator and identified by an OX_ID.

originatorExchangeId The Originator Exchange_ID (OX_ID) is a two-byte field (Word 4, Bits 31-16) that identifies the Exchange_ID assigned by the Originator of the Exchange.

parameter The Parameter type has meanings based on frame type. For Link_Control frames, the Parameter type is used to carry information specific to the individual Link_Control frame. For Data frames with the relative offset present bit set to 1, the Parameter type specifies relative offset. For Data frames with the relative offset Present bit set to zero, the Parameter type is set and interpreted in a protocol specific manner that may depend on the type of Information Unit carried by the frame.

EXAMPLES See under *fibreChannel*.

SEE ALSO *fibreChannel*

NAME - espHeader

espHeader-inserts Encapsulating Security Payload (ESP) header in a fibre channel packet

SYNOPSIS

espHeader sub-command options

DESCRIPTION

Encapsulating Security Payload (ESP) is a generic mechanism to provide confidentiality, data origin authentication, and anti-replay protection to IP packets. ESP is applied to Fibre Channel frames in transport mode.

STANDARD OPTIONS

sequenceNumber It is an unsigned 32-bit field that contains a counter value that increases by one for each packet sent, as per the source address packet sequence number.

securityParameter Index It is a 32-bit value that is used by a receiver to identify the source address to which an incoming packet is bound. The SPI field is mandatory in an ESP header.

EXAMPLES

See under *fibreChannel*.

SEE ALSO

fibreChannel

NAME - extendedLinkServices

extendedLinkServices-inserts Extended Link Services (ELS) protocols in a fibre channel module

SYNOPSIS

extendedLinkServices *sub-command options*

DESCRIPTION

An Extended Link Service (ELS) request solicits a destination Nx_Port to perform a function. An ELS reply is transmitted in response to an ELS request, unless otherwise specified. Each request or reply is composed of a single Sequence with the ELS_Command code being specified in the first word of the Payload of the first frame of the Sequence.

STANDARD OPTIONS

fcElsProtocolType

The Extended Link Services protocol types are as follows:

Option	Usage
<i>ElsFlogi</i>	Sets Fabric Login (FLOGI) ELS Service Parameters.
<i>ElsPlogi</i>	Sets Process Login (PLOGI) ELS Service Parameters.
<i>ElsFdisc</i>	Sets the Discover F_Port Service (FDISC) ELS Service Parameters.
<i>ElsLsAcc</i>	Sets the Link Service Accept (LS_ACC) ELS Service Parameters.
<i>ElsLogo</i>	Sets the Link Service Accept (LS_ACC) ELS Service Parameters.
<i>ElsScr</i>	Sets the State Change Registration (SCR) ELS Service Parameters.
<i>ElsLsRjt</i>	Sets the Link Service Reject (LS_RJT) ELS Service Parameters.
<i>ElsRscn</i>	Sets the Registered State Change Notification (RSCN) ELS Service Parameters.

elsFlogi

The Fabric Login (FLOGI) ELS transfers Service Parameters from the initiating Nx_Port to the FC_Port associated with the D_ID. The FLOGI frame provides the means by which an Nx_Port requests Login with the Fabric. Login with the Fabric is required for all Nx_Ports, regardless of the class supported. Communication with other Nx_Ports is not attempted until the Fabric Login procedure is complete.

The options are as follows:

Option	Usage
<i>bufferToBufferCredit</i>	It is the limiting value for BB_Credit_CNT in the buffer-to-buffer flow control model. If a Fabric is present, FLOGI initializes the buffer-to-buffer Credit.

Option	Usage
<i>bbScNumber</i>	The Buffer-to-buffer State Change Number (BB_SC_N) field specifies the Buffer-to-buffer State Change Number. It indicates that the sender of the FLOGI frame is requesting 2BB_SC_N number of frames to be sent between two consecutive BB_SCs primitives, and 2BB_SC_N number of R_RDY primitives to be sent between two consecutive BB_SCr primitives.
<i>receiveDataFieldSize</i>	The field size of the data received from the FC_Port.
<i>portWWN</i>	The eight-byte field that identifies an FC_Port.
<i>nodeWWN</i>	The eight-byte name identifier associated with a node.
<i>eDTOV</i>	The EDTOV value.

elsPlogi

The PLOGI ELS transfers Service Parameters from the initiating Nx_Port to the FC_Port associated with the D_ID. The PLOGI frame provides the means by which an Nx_Port requests Login with another Nx_Port prior to other Data frame transfers.

The options are as follows:

Option	Usage
<i>bufferToBufferCredit</i>	It is the limiting value for BB_Credit_CNT in the buffer-to-buffer flow control model. If a Fabric is present, FLOGI initializes the buffer-to-buffer Credit.
<i>bbScNumber</i>	The Buffer-to-buffer State Change Number (BB_SC_N) field specifies the Buffer-to-buffer State Change Number. It indicates that the sender of the PLOGI frame is requesting 2BB_SC_N number of frames to be sent between two consecutive BB_SCs primitives, and 2BB_SC_N number of R_RDY primitives to be sent between two consecutive BB_SCr primitives.
<i>receiveDataFieldSize</i>	The field size of the data received from the FC_Port.
<i>portWWN</i>	The eight-byte field that identifies an FC_Port.
<i>nodeWWN</i>	The eight-byte name identifier associated with a node.
<i>eDTOV</i>	The EDTOV value.

elsFdisc

The Discover F_Port Service Parameters (FDISC) ELS transfers Service Parameters from the initiating Nx_Port to the Fx_Port at well-known F_Port_ID. This provides the means for the exchange of Service Parameters and the assignment of an additional N_Port_IDs without changing service parameters.

The options are as follows:

Option	Usage
<i>bufferToBufferCredit</i>	It is the limiting value for BB_Credit_CNT in the buffer-to-buffer flow control model. If a Fabric is present, FLOGI initializes the buffer-to-buffer Credit.
<i>bbScNumber</i>	The Buffer-to-buffer State Change Number (BB_SC_N) field specifies the Buffer-to-buffer State Change Number. It indicates that the sender of the PLOGI frame is requesting 2BB_SC_N number of frames to be sent between two consecutive BB_SCs primitives, and 2BB_SC_N number of R_RDY primitives to be sent between two consecutive BB_SCr primitives.
<i>receiveDataFieldSize</i>	The field size of the data received from the FC_Port.
<i>portWWN</i>	The eight-byte field that identifies an FC_Port.
<i>nodeWWN</i>	The eight-byte name identifier associated with a node.
<i>eDTOV</i>	The EDTOV value.

eIsLsAcc

The Link Service Accept (LS_ACC) ELS reply Sequence notifies the originator of an ELS request that the ELS request Sequence has been completed. The Responder terminates the Exchange by setting the Last Sequence bit (Bit 20) in F_CTL on the last Data frame of the reply Sequence. The first byte of the Payload contains 02h. The remainder of the Payload is unique to the ELS request.

The options are as follows:

Option	Usage
<i>bufferToBufferCredit</i>	It is the limiting value for BB_Credit_CNT in the buffer-to-buffer flow control model. If a Fabric is present, FLOGI initializes the buffer-to-buffer Credit.
<i>bbScNumber</i>	The Buffer-to-buffer State Change Number (BB_SC_N) field specifies the Buffer-to-buffer State Change Number. It indicates that the sender of the PLOGI frame is requesting 2BB_SC_N number of frames to be sent between two consecutive BB_SCs primitives, and 2BB_SC_N number of R_RDY primitives to be sent between two consecutive BB_SCr primitives.
<i>receiveDataFieldSize</i>	The field size of the data received from the FC_Port.
<i>portWWN</i>	The eight-byte field that identifies an FC_Port.
<i>nodeWWN</i>	The eight-byte name identifier associated with a node.
<i>eDTOV</i>	The EDTOV value.

elsLogo

The LOGO ELS provides a method for explicitly removing service between two Nx_Port_IDs or between an N_Port_ID and a Fabric. Logout releases resources, identifiers, and relationships associated with maintaining service between an Nx_Port_ID and a destination Nx_Port_ID or Fabric.

The options are as follows:

Option	Usage
<i>portId</i>	The unique address identifier of the FC Port.
<i>portName</i>	The eight-byte field that identifies the FC Port.

elsScr

The State Change Registration (SCR) ELS requests the Fabric Controller or Nx_Port to add the Nx_Port that is sending the SCR Request to the list of Nx_Ports registered to receive the RSCN ELS.

The options are as follows:

Option	Usage
<i>registrationFunction</i>	The Registration Functions for SCR.

fcElsScrRegFunction

The Registration Functions for SCR ELS.

The options are as follows:

Option	Usage
<i>elsScrReserved</i>	The reserved format with value 0.
<i>elsScrFabricDetectedRegistration</i>	Register to receive all RSCN Requests issued by the Fabric Controller for events detected by the Fabric.
<i>elsScrNxPortDetectedRegistration</i>	Register to receive all RSCN Requests issued for events detected by the affected Nx_Port.
<i>elsScrFullRegistration</i>	Register to receive all RSCN Requests issued. The RSCN Request returns all affected N_Port_ID pages.
<i>elsScrClearRegistration</i>	Removes any current RSCN registrations.

elsLsRjt

The Link Service Reject (LS_RJT) notifies the transmitter of a Link Service request that the Link Service request Sequence has been rejected. A four-byte reason code is contained in the Data Field. Link Service Reject is transmitted for a variety of conditions that are unique to a specific Link Service request. For example, if the Service Parameters specified in a Login frame were logically inconsistent or in error, a P_RJT frame would not be transmitted in response, but rather a Link Service Reject.

The options are as follows:

Option	Usage
<i>FcElsRjtReasonCode</i>	The ELS LS_RJT reason codes.

FcElsRjtReasonCode

The ELS LS_RJT reason codes.

The options are as follows:

Option	Usage
<code>elsRjtInvalidELSCommandcode</code>	The ELS_Command code in the Sequence being rejected is invalid.
<code>elsRjtLogicalError</code>	The request identified by the ELS_Command code and Payload content is invalid or logically inconsistent for the conditions present.
<code>elsRjtLogicalbusy</code>	The Link Service is logically busy and unable to process the request at this time.
<code>elsRjtProtocolError</code>	This indicates that an error has been detected that violates the rules of the ELS Protocol that are not specified by other error codes.
<code>elsRjtUnableToPerformCommand</code>	The Recipient of a Link Service command is unable to perform the request at this time.
<code>elsRjtCommandNotSupported</code>	The Recipient of a Link Service command does not support the command requested.
<code>elsRjtCommandAlreadyInProgress</code>	The command progress is tracked.
<code>elsRjtVendorSpecificError</code>	The Vendor specific error bits may be used by Vendors to specify additional reason codes.

elsRscn

The Registered State Change Notification (RSCN) ELS is sent to registered Nx_Ports when an event occurs that may have affected the state of one or more Nx_Ports, or the ULP state within the Nx_Port. The term, state, is used here to refer to any condition of an Nx_Port that is considered important enough to notify other Nx_Ports of a change in that state. The RSCN provides an indication of the change of state that is being reported.

The options are as follows:

Option	Usage
<code>pageLength</code>	The length in bytes of an affected Port_ID page. This value is fixed at 04h.
<code>payLoadLength</code>	The length in bytes of the entire Payload, inclusive of the word 0. This value is a multiple of 4 bytes. The minimum value of this field is 8 bytes. The maximum value of this field is 1024 bytes.

COMMANDS

The `extendedLinkServices` command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

extendedLinkServices **setDefault** *option*

Sets to IxTclHal default values for all configuration options.

extendedLinkServices **set** *option*

Sets the current configuration of the `extendedLinkServices` for the indicated port. Call this command before calling `extendedLinkServices get option value` to get the value of the configuration option.

extendedLinkServices *get option*

Gets the current configuration of the *extendedLinkServices* for the indicated port.

EXAMPLES

See under *fibreChannel*.

SEE ALSO

fibreChannel

NAME - fcEOF

fcEOF-insert Fibre Channel End-of-Frame (EOF) delimiter. It is an Ordered Set that immediately follows the CRC.

SYNOPSIS	fcEOF <i>sub-command options</i>
-----------------	----------------------------------

DESCRIPTION	The End-of-Frame (EOF) delimiter is an Ordered Set that immediately precedes the frame content.
--------------------	---

STANDARD OPTIONS

fcEOFDelimiter	The multiple EOF delimiters defined for Sequence control are as follows:
-----------------------	--

Option	Usage
<i>fcEOFt</i>	The EOFt indicates that the Sequence associated with this SEQ_ID is complete. EOFt or EOFdt is used to properly close a Sequence without error.
<i>fcEOFdt</i>	EOFdt is used to properly close a Sequence without error.
<i>fcEOFa</i>	The EOFa terminates a partial frame due to a malfunction in a link facility during transmission.
<i>fcEOFn</i>	The EOFn identifies the end of frame when one of the other EOF delimiters indicating valid frame content is not required.
<i>fcEOFni</i>	EOFni replaces an EOFn or EOFt, indicating that the frame content is invalid.
<i>fcEOFdti</i>	EOFdti is used to properly close a Sequence without error.
<i>fcEOFrt</i>	The EOFrt removes a dedicated connection through a Fabric. The connection is removed and terminated.
<i>fcEOFrti</i>	Remove Terminate Invalid: The EOFrti replaces a recognized EOFrt delimiter on a frame of invalid frame content.

COMMANDS	The <i>fcEOF</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

fcEOF setDefault *option*

Sets to IxTclHal default values for all configuration options.

fcEOF set *option*

Sets the current configuration of the *fcEOF* for the indicated port. Call this command before calling *fcEOF get option value* to get the value of the configuration option.

fcEOF get option

Gets the current configuration of the *fcEOF* for the indicated port.

EXAMPLES

See under *fibreChannel*.

SEE ALSO

fibreChannel

NAME - fcNameServer

fcNameServer-The FC Name Server command enables to setup the configurable parameters for the Name Server.

SYNOPSIS	fcNameServer <i>sub-command options</i>
DESCRIPTION	The <i>fcNameServer</i> command is used to send name server queries to the Fibre Channel module.
STANDARD OPTIONS	
enableRnnId <i>true/false</i>	The RNN_ID Name Server request is used to associate a Node Name with a given Port Identifier (<i>default = true</i>).
enableRcsId <i>true/false</i>	The RCS_ID Name Server request is used to record the Classes of Service that are supported by a given Port Identifier (<i>default = false</i>).
enableRftId <i>true/false</i>	The RFT_ID Name Server request is used to record the FC-4 TYPES that are supported by a given Port Identifier (<i>default = true</i>).
enableRpnId <i>true/false</i>	The RPN_ID Name Server request is used to record the Port Name that is supported by a given Port Identifier (<i>default = false</i>).
enableRptId <i>true/false</i>	The RPT_ID Name Server request is used to record the Port Type that is supported by a given Port Identifier (<i>default = false</i>).
enableRspnId <i>true/false</i>	The RSPN_ID Name Server request is used to associate a Symbolic Port Name with a given Port Identifier (<i>default = false</i>).
enableRsnnNm <i>true/false</i>	The RSNN_NN Name Server request is used to associate a Symbolic Node Name with a given Node Name (<i>default = false</i>).
enableRhaId <i>true/false</i>	The RHA_ID Name Server request is used to associate a Hard Address with a given Port Identifier (<i>default = false</i>).
symbolicPortName	A user-defined string to identify a port, for example 'Ixia Port 1'.
symbolicNodeName	A user-defined string to identify a node, for example 'Ixia Node 1'.
COMMANDS	
	The <i>fcNameServer</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

fcNameServer **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **fcNameServer** command.

fcNameServer config *option value*

Modify the fcNameServer configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

fcNameServer setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *fibreChannel*.

SEE ALSO

fibreChannel

NAME - fcNameServerQuery

fcNameServerQuery-The FC Name Server Query sends name server queries to the Fibre Channel module.

SYNOPSIS	fcNameServerQuery <i>sub-command options</i>
-----------------	--

DESCRIPTION	The <i>fcNameServerQuery</i> command is used to send name server queries to the Fibre Channel module.
--------------------	---

STANDARD OPTIONS

fcNameServerQuery Command	Allows to set the name server queries for Fibre Channel. The type of commands are as follows:
----------------------------------	---

Option	Value	Usage
<i>commandGANxt</i>	256	The GA_NXT is used by a requestor to obtain Name Server objects associated with a specific Port.
<i>commandGIDA</i>	257	When the Name Server receives a GID_A request, it returns identifiers for the specified scope.
<i>commandGPNId</i>	274	When the Name Server receives a GPN_ID request, it returns the registered Port Name object for the specified Port Identifier.
<i>commandGNNId</i>	275	When the Name Server receives a GNN_ID request, it returns the registered Node Name object for the specified Port Identifier.
<i>commandGIDPn</i>	299	When the Name Server receives a GID_PN request, it returns the Port Identifier associated with the specified Port Name.
<i>commandGIDPt</i>	417	When the Name Server receives a GID_PT request, it returns all Port Identifiers having registered support for the specified Port Type. If the specified Port Type is equal to 'Nx_Port', then the Name Server returns all Port Identifiers that have registered Port Types with an unsigned value of less than 80h.

fcNameServerQuery Object	Depends on the query command code. The types of name server query objects are as follows:
---------------------------------	---

Option	Value	Usage
<i>objectPortId</i>	1	The Port Identifier.
<i>objectPortName</i>	2	Indicates the port name.
<i>objectPortType</i>	3	Indicates the port type.
<i>objectNone</i>	0	

COMMANDS

The *fcNameServerQuery* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

fcNameServerQuery set

Sets the current configuration of the *fcNameServerQuery* for the indicated port. Call this command before calling *fcNameServerQuery get option value* to get the value of the configuration option.

fcNameServerQuery get

Gets the current configuration of the *fcNameServerQuery* for the indicated port.

fcNameServerQuery setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *fibreChannel*.

SEE ALSO

fibreChannel

NAME - fcoe

fcoe - configure Fibre Channel over Ethernet header and trailer packet.

SYNOPSIS	fcoe <i>sub-command options</i>																														
DESCRIPTION	The <i>fcoe</i> command is used to configure Fibre Channel over Ethernet (FCoE) header and trailer packet. FCoE is a method of communicating data for streams and protocols.																														
STANDARD OPTIONS																															
eEofDelimiter	Configure the end of frame delimiter. (<i>default = 65</i>) Available options are:																														
<table border="1"> <thead> <tr> <th>Option</th><th>Value</th><th>Usage</th></tr> </thead> <tbody> <tr> <td>fcoeEofTerminate</td><td>66</td><td>(default) End of frame terminate</td></tr> <tr> <td>fcoeEofAbort</td><td>80</td><td>EoF abort</td></tr> <tr> <td>fcoeEofNormal</td><td>65</td><td>EoF normal</td></tr> <tr> <td>fcoeEofNormalInvalid</td><td>73</td><td>EoF normal invalid</td></tr> <tr> <td>fcoeEofRemoveTerminateClass4</td><td>68</td><td>EoF remove terminate class 4</td></tr> <tr> <td>fcoeEofRemoveTerminateInvalidClass4</td><td>79</td><td>EoF remove terminate invalid class 4</td></tr> </tbody> </table>		Option	Value	Usage	fcoeEofTerminate	66	(default) End of frame terminate	fcoeEofAbort	80	EoF abort	fcoeEofNormal	65	EoF normal	fcoeEofNormalInvalid	73	EoF normal invalid	fcoeEofRemoveTerminateClass4	68	EoF remove terminate class 4	fcoeEofRemoveTerminateInvalidClass4	79	EoF remove terminate invalid class 4									
Option	Value	Usage																													
fcoeEofTerminate	66	(default) End of frame terminate																													
fcoeEofAbort	80	EoF abort																													
fcoeEofNormal	65	EoF normal																													
fcoeEofNormalInvalid	73	EoF normal invalid																													
fcoeEofRemoveTerminateClass4	68	EoF remove terminate class 4																													
fcoeEofRemoveTerminateInvalidClass4	79	EoF remove terminate invalid class 4																													
eEofReserved	Configure the end of frame reserved value. (<i>default = '00 00 00</i>)																														
enableValidateFrame Size true/false	Enable the stream size validation. The frame size should be a multiple of 4. (<i>default = false</i>)																														
eSofReserved	Configure the start of frame reserved value. This is a 12-byte hex value. (<i>default = '00 .. 00</i>)																														
eSofDelimiter	Configure the start of frame delimiter. (<i>default = 54</i>) Available option values are:																														
<table border="1"> <thead> <tr> <th>Option</th><th>Value</th><th>Usage</th></tr> </thead> <tbody> <tr> <td>fcoeSofNormalClass1</td><td>55</td><td>Start of frame normal class 1</td></tr> <tr> <td>fcoeSofInitiateClass2</td><td>45</td><td>SoF initiate class 2</td></tr> <tr> <td>fcoeSofNormalClass2</td><td>53</td><td>SoF normal class 2</td></tr> <tr> <td>fcoeSofInitiateClass3</td><td>46</td><td>SoF initiate class 3</td></tr> <tr> <td>fcoeSofNormalClass3</td><td>54</td><td>(default) SoF normal class 3</td></tr> <tr> <td>fcoeSofActivateClass4</td><td>57</td><td>SoF activate class 4</td></tr> <tr> <td>fcoeSofInitiateClass4</td><td>41</td><td>SoF initiate class 4</td></tr> <tr> <td>fcoeSofNormalClass4</td><td>49</td><td>SoF normal class 4</td></tr> <tr> <td>fcoeSofFabric</td><td>40</td><td>SoF fabric</td></tr> </tbody> </table>		Option	Value	Usage	fcoeSofNormalClass1	55	Start of frame normal class 1	fcoeSofInitiateClass2	45	SoF initiate class 2	fcoeSofNormalClass2	53	SoF normal class 2	fcoeSofInitiateClass3	46	SoF initiate class 3	fcoeSofNormalClass3	54	(default) SoF normal class 3	fcoeSofActivateClass4	57	SoF activate class 4	fcoeSofInitiateClass4	41	SoF initiate class 4	fcoeSofNormalClass4	49	SoF normal class 4	fcoeSofFabric	40	SoF fabric
Option	Value	Usage																													
fcoeSofNormalClass1	55	Start of frame normal class 1																													
fcoeSofInitiateClass2	45	SoF initiate class 2																													
fcoeSofNormalClass2	53	SoF normal class 2																													
fcoeSofInitiateClass3	46	SoF initiate class 3																													
fcoeSofNormalClass3	54	(default) SoF normal class 3																													
fcoeSofActivateClass4	57	SoF activate class 4																													
fcoeSofInitiateClass4	41	SoF initiate class 4																													
fcoeSofNormalClass4	49	SoF normal class 4																													
fcoeSofFabric	40	SoF fabric																													
version	Configure the version. (<i>default = 1</i>)																														
COMMANDS	The <i>fcoe</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.																														

fcoe decode capFrame chasID cardID portID

Decodes the FCoE header and trailer packet and refreshes the IxTclHal object.
Specific errors are:

- No connection to a chassis
- The captured frame is not a valid fcoe packet

fcoe get chasID cardID portID

Gets current FCoE header and trailer settings from IxHal and refreshes IxTclHal object. Specific errors are:

- No connection to a chassis
- Invalid port

fcoe set chasID cardID portID

Sets current FCoE header and trailer settings from IxTclHal to local IxHal.
Specific errors are:

- No connection to a chassis
- Unsupported feature
- The port is being used by another user
- The configured parameters are not valid for this port

fcoe setDefault chasID cardID portID

Sets to IxTclHal local default.

EXAMPLES

```

package req IxTclHal

set hostnameixia_hostname

if {[ixConnectToChassis $hostname]} {
    errorMsg "error connecting $hostname chassis"
    return "FAIL"
}

set chassId      [chassis cget -id]
set cardId       2
set portId       1
set streamId1
set portList [list [list $chassId $cardId $portId] ]

if {! [port isValidFeature $chassId $cardId $portId
$::portFeatureDataCenterMode]} {
    errorMsg "portFeatureDataCenterMode is not valid on
$chassId $cardId $portId"
    return "FAIL"
}

# Configure FCoE interfaces
proc configurePortAndProtocols { portList } \
{
    set retCode $::TCL_OK

    if {[ixSetWidePacketGroupMode portList]} {
        errorMsg "Error in Setting Wide Packet Group Mode"
        set retCode $::TCL_ERROR
    }
}

```

```

foreach port $portList {
    scan $port "%d %d %d" chassisId cardId portId

    port setFactoryDefaults $chassisId $cardId $portId
    port config -flowControl ::true
    port config -enableAutoDetectInstrumentation
    ::true
    port config -autoDetectInstrumentationMode
    ::portAutoInstrumentationModeFloating
    port config -loopback ::portLoopback
    port config -receiveMode [expr
    ::portCapture|::portRxSequenceChecking|::portRxModeWidePacketG
    roup]
    port config -transmitMode
    ::portTxModeAdvancedScheduler
    if {[port set $chassisId $cardId $portId]} {
        errorMsg "Error calling port set $chassisId
$cardId $portId"
        set retCode ::TCL_ERROR
    }

    stat setDefault
    stat config -mode statNormal
    stat config -enableFcoeStats ::true
    stat config -fcoeRxSharedStatType1
    ::statFcoeValidFrames
    stat config -fcoeRxSharedStatType2
    ::statFcoeValidFrames
    if {[stat set $chassisId $cardId $portId]} {
        errorMsg "Error calling stat set $chassisId
$cardId $portId"
        set retCode ::TCL_ERROR
    }

    packetGroup setDefault
    packetGroup config -enableInsertPgid true
    packetGroup config -latencyControl cutThrough
    packetGroup config -groupIdMode packetGroupSplit
    if {[packetGroup setRx $chassisId $cardId $portId]}
    {
        errorMsg "Error calling packetGroup setRx
$chassisId $cardId $portId"
        set retCode ::TCL_ERROR
    }

    splitPacketGroup setDefault
    splitPacketGroup config -groupIdOffset 32
    splitPacketGroup config -groupIdOffsetBaseType
    ::splitPgidStartOfFrame
    splitPacketGroup config -groupIdWidth 4
    splitPacketGroup config -groupIdMask "FF FF 00 00"
    if {[splitPacketGroup set $chassisId $cardId
    $portId 0]} {
        errorMsg "Error calling splitPacketGroup
set $chassisId $cardId $portId 0"
        set retCode ::TCL_ERROR
    }

    splitPacketGroup setDefault
    splitPacketGroup config -groupIdOffset 52

```

```

        splitPacketGroup config -groupIdOffsetBaseType
$:::splitPgidStartOfFrame
        splitPacketGroup config -groupIdWidth 4
        splitPacketGroup config -groupIdMask "FF FF FF FF"
        if {[splitPacketGroup set $chassId $cardId
$portId 1]} {
            errorMsg "Error calling splitPacketGroup
set $chassId $cardId $portId 1"
            set retCode $:::TCL_ERROR
        }

        splitPacketGroup setDefault
        splitPacketGroup config -groupIdOffset 52
        splitPacketGroup config -groupIdOffsetBaseType
$:::splitPgidStartOfFrame
        splitPacketGroup config -groupIdWidth 4
        splitPacketGroup config -groupIdMask
"FF FF FF FF"
        if {[splitPacketGroup set $chassId $cardId
$portId 2]} {
            errorMsg "Error calling splitPacketGroup
set $chassId $cardId $portId 2"
            set retCode $:::TCL_ERROR
        }

        autoDetectInstrumentation setDefault
        autoDetectInstrumentation config -startOfScan 0
        autoDetectInstrumentation config -signature
{87 73 67 49 42 87 11 80 08 71 18 05}
        if {[autoDetectInstrumentation setRx $chassId
$cardId $portId]} {
            errorMsg "Error calling
autoDetectInstrumentation setRx $chassId $cardId $portId"
            set retCode $:::TCL_ERROR
        }

        if {[interfaceTable select $chassId $cardId
$portId]} {
            errorMsg "Error calling interfaceTable
select $chassId $cardId $portId"
            set retCode $:::TCL_ERROR
        }

        interfaceTable setDefault
        interfaceTable config -fcoeRequestRate 500
        interfaceTable config -fcoeNumRetries 5
        if {[interfaceTable set]} {
            errorMsg "Error calling interfaceTable
set"
            set retCode $:::TCL_ERROR
        }

        interfaceTable clearAllInterfaces

        ##### Interface entry type -
interfaceTypeConnected

        interfaceEntry clearAllItems addressTypeIpV6
        interfaceEntry clearAllItems addressTypeIpV4
        interfaceEntry setDefault

        fcoeProperties setDefault

```

```

        fcoeProperties config -sourcePortWWN
"02 00 04 FF FE 9F 0A 5C"
        fcoeProperties config -sourceNodeWWN
"02 00 04 00 00 9F 0A 5C"
        fcoeProperties config -destinationId "01.b6.69"
        fcoeProperties config -sourceOui "0e.fc.00"
        fcoeProperties config -bufferToBufferRxSize 2112
        fcoeProperties config -enableNs $::false
        fcoeProperties config -enablePlogi $::false

        interfaceEntry config -enable true
        interfaceEntry config -description
"ProtocolInterface1"
        interfaceEntry config -macAddress
{00 00 04 9F 0A 5C}
        interfaceEntry config -eui64Id
{02 00 04 FF FE 9F 0A 5C}
        interfaceEntry config -mtu 1500
        interfaceEntry config -enableFlogi $::true
        if {[interfaceTable addInterface
interfaceTypeConnected]} {
            errorMsg "Error calling interfaceTable
addInterface interfaceTypeConnected"
            set retCode $::TCL_ERROR
        }

##### Interface entry type - interfaceTypeNpiv

        interfaceEntry clearAllItems addressTypeIPv6
        interfaceEntry clearAllItems addressTypeIPv4
        interfaceEntry setDefault

        npivProperties setDefault
        npivProperties config -sourcePortWWN
"02 00 04 FF FE 9F 0A 5D"
        npivProperties config -sourceNodeWWN
"02 00 04 00 00 9F 0A 5D"
        npivProperties config -destinationId "01.b6.69"
        npivProperties config -bufferToBufferRxSize 2112
        npivProperties config -enableNs $::false
        npivProperties config -enablePlogi $::false

        interfaceEntry config -enable $::true
        interfaceEntry config -description
"NpivInterface2"
        interfaceEntry config -connectedVia
"ProtocolInterface1"
        if {[interfaceTable addInterface
interfaceTypeNpiv]} {
            errorMsg "Error calling interfaceTable
addInterface interfaceTypeNpiv"
            set retCode $::TCL_ERROR
        }

}

if {[ixWritePortsToHardware portList]} {
    errorMsg "Error ixWritePortsToHardware"
    set retCode $::TCL_ERROR
}

if {[ixCheckLinkState portList]} {
    errorMsg "Error ixCheckLinkState"
}

```

```

        set retCode $::TCL_ERROR
    }

    return $retCode
}

#Configure FCoE streams
proc configureFcoeStreams { portList SourceIdArray } \
{
    upvar $SourceIdArray sourceIdArray

    set retCode $::TCL_OK

    foreach port $portList {
        scan $port "%d %d %d" chassisId cardId portId
        port reset $chassisId $cardId $portId
        for {set streamId 1 } {$streamId <= 2 } { incr
streamId} {

            # Stream 1
            protocol setDefault
            protocol config -name $::fcoe
            protocol config -ethernetType
$::etherenetII

            stream setDefault
            stream config -enable $::true
            stream config -numFrames 1000
            stream config -gapUnit $::gapNanoSeconds
            stream config -rateMode $::usePercentRate
            stream config -framesize 100
            stream config -frameSizeType $::sizeFixed
            stream config -patternType $::incrByte
            stream config -dataPattern x00010203
            stream config -pattern "00 01 02 03"
            stream config -frameType "89 06"
            stream config -dma $::stopStream
            stream config -enableStatistic $::true
            stream config -enableSourceInterface
$::true

            stream config -priorityGroup
$::priorityGroup0
            stream config -patternType $::incrByte
            stream config -preambleSize 8

            if {$streamId == 1} {
                stream config -name "FCoE stream"
                stream config -sa
                stream config -da
                stream config -percentPacketRate
50
                stream config -
sourceInterfaceDescription "ProtocolInterface1"
            } else {
                stream config -name "NPIV stream"
                stream config -sa
                stream config -da
            }
        }
    }
}

```

```

streamconfig-percentPacketRate
50.000014565

sourceInterfaceDescription      stream config -
"sourceInterfaceDescription"    "NpivInterface2"
}

fcoe setDefault
fcoe config -enableValidateFrameSize
$::true
fcoe config -version 1
fcoe config -eSofReserved
"00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00"
fcoe config -eSofDelimiter
$::fcoeSofNormalClass3
fcoe config -eEofDelimiter
$::fcoeEofTerminate
fcoe config -eEofReserved "00 00 00"

if {[fcoe set $chassisId $cardId $portId]}
{
    errorMsg "Error calling fcoe set
$chassisId $cardId $portId"
    set retCode $::TCL_ERROR
}

fibreChannel setDefault
fibreChannel config -destinationId
"00.00.00"
fibreChannel config -routingControlType
$::fibreChannelDeviceDataFrames
fibreChannel config -
routingControlInformation
$::fibreChannelUncategorizedInformation
    if {$streamId == 1} {
        fibreChannel config -sourceId
$sourceIdArray($chassisId,$cardId,$portId,fcoe)
    } else {
        fibreChannel config -sourceId
$sourceIdArray($chassisId,$cardId,$portId,npiv)
    }

fibreChannel config -
csControlOrPriorityValue          0x00
    fibreChannel config -frameControl "00 00 00"
    fibreChannel config -type 0x00
    fibreChannel config -sequenceCount 5
    fibreChannel config -dataFieldControl 0x00
    fibreChannel config -sequenceId 0x00
    fibreChannel config -responderExchangeId "00 00"
    fibreChannel config -originatorExchangeId "00 00"
    fibreChannel config -parameter "00 00 00 00"
    fibreChannel config -originatorExchangeCounter
fibreChannelIdle
    fibreChannel config -enableBadFibreChannelCrc
$::true
    fibreChannel config -enableUseFcControlBits
$::true
    fibreChannel config -exchangeContext
$::fibreChannelOriginator
    fibreChannel config -sequenceContext
$::fibreChannelInitiator

```

```

        fibreChannel config -firstSequence
$::fibreChannelFirstSequenceOther
        fibreChannel config -lastSequence
$::fibreChannelLastSequenceOther
        fibreChannel config -endSequence
$::fibreChannelEndSequenceOther
        fibreChannel config -endConnection
$::fibreChannelConnectionAlive
        fibreChannel config -csControlOrPriority
$::fibreChannelCsCtrl
        fibreChannel config -sequenceInitiative
$::fibreChannelInitiativeHold
        fibreChannel config -ackForm
$::fibreChannelOriginal
        fibreChannel config -
retransmittedSequence      $::fibreChannelOriginal
        fibreChannel config -
unidirectionalTransmit    $::fibreChannelBidirectional
        fibreChannel config -
continueSequenceCondition $::fibreChannelNoInformation
        fibreChannel config -
abortSequenceCondition    $::fibreChannelContinue
        fibreChannel config -
relativeOffsetPresent
$::fibreChannelRelativeOffsetDefined
        fibreChannel config -exchangeReassembly
$::fibreChannelExchangeReassemblyOff
        fibreChannel config -fillBytes
$::fibreChannelZeroHexByteFill
        if {[fibreChannel set $chassId $cardId
$portId] } {
            errorMsg "Error calling
fibreChannel set $chassId $cardId $portId"
            set retCode $::TCL_ERROR
        }
        if {[stream set $chassId $cardId $portId
$streamId] } {
            errorMsg "Error calling stream
set $chassId $cardId $portId $streamId"
            set retCode $::TCL_ERROR
        }
        packetGroup setDefault
        packetGroup config -signature
"08 71 18 05"
        packetGroup config -insertSignature
$::true
        packetGroup config -groupId 1
        packetGroup config -groupIdOffset 66
        packetGroup config -enableInsertPgid
$::true
        packetGroup config -sequenceNumberOffset
68
        packetGroup config -
                2
        packetGroup config -
                $::true
        packetGroup config -latencyControl
$::cutThrough
        if {[packetGroup setTx $chassId $cardId
$portId $streamId] } {

```

```
                errorMessage "Error calling
packetGroup setTx $chassisId $cardId $portId $streamId"
                           set retCode $::TCL_ERROR
}
}

dataIntegrity setDefault
dataIntegrity config -signatureOffset 52
dataIntegrity config -signature
"08 71 18 00"
dataIntegrity config -insertSignature
$::true
dataIntegrity config -enableTimeStamp
$::false
dataIntegrity config -
floatingTimestampAndDataIntegrityMode
$::dataIntegrityNumberOfBytesFromEndOfFrame
dataIntegrity config -
numBytesFromEndOfFrame      12
dataIntegrity config -payloadLength 0
if {[dataIntegrity setTx $chassisId $cardId
$portId $streamId]} {
    errorMessage "Error calling
dataIntegrity setTx $chassisId $cardId $portId $streamId"
    set retCode $::TCL_ERROR
}

autoDetectInstrumentation setDefault
autoDetectInstrumentation config -
enableTxAutomaticInstrumentation   $::true
autoDetectInstrumentation config -
signature {87 73 67 49 42 87 11 80 08 71 18 05}
if {[autoDetectInstrumentation setTx
$chassisId $cardId $portId $streamId]} {
    errorMessage "Error calling
autoDetectInstrumentation setTx $chassisId $cardId $portId
$streamId"
    set retCode $::TCL_ERROR
}
}

ixWriteConfigToHardware portList
return $retCode
}

proc fcoeMainTest { portList } \
{
    errorMessage "***** Testing Latency Test and Stream-
Interface linkage Test on $portList"

    set retCode $::TCL_OK

    if {[configurePortAndProtocols $portList ]} {
        errorMessage "Error configurePortAndProtocols"
        set retCode $::TCL_ERROR
    }

    errorMessage "Starting FCoE Server..."
    package require IxTclServices
    set pcuCommand "/shared/chassis/arch/bin/fcoeserver&"
    if {[issuePcpuCommand portList $pcuCommand]} {
        errorMessage "Failed to start FCoE server"
        set retCode $::TCL_ERROR
    }
}
```

```

        } else {
            errorMsg "FCoE Server started..."
        }

# Give some time for FCoE server to start
after 4000

# Verify FCoE discovered information

foreach port $portList {
    scan $port "%d %d %d" chassisId cardId portId

    if {[interfaceTable select $chassisId $cardId
$portId]} {
        errorMsg "Error selecting interfaceTable
on $chassisId $cardId $portId."
        set retCode $::TCL_ERROR
    }

    if {[interfaceTable requestDiscoveredTable]} {
        errorMsg "Error interfaceTable
requestDiscoveredTable on $chassisId $cardId $portId."
        set retCode $::TCL_ERROR
    }

    if {[interfaceTable getFirstInterface
interfaceTypeConnected ]} {
        errorMsg "Error adding
interfaceTypeConnected to interfaceTable on $chassisId $cardId
$portId."
        set retCode $::TCL_ERROR
    }

    set interfaceDescription [interfaceEntry cget -
description]

    after 2000

    fcoeDiscoveredInfo setDefault
    if {[interfaceTable getFcoeDiscoveredInfo
$interfaceDescription]} {
        errorMsg "Error getting Fcoe Discovered
table for $interfaceDescription on $chassisId $cardId $portId."
        set retCode $::TCL_ERROR
    }
    set sourceIdArray($chassisId,$cardId,$portId,fcoe)
[fcoeDiscoveredInfo cget -sourceId]
    ixPuts ">>>> $interfaceDescription
DiscoveredInfo [fcoeDiscoveredInfo cget -sourceId]"

    set pgidStringFcoe      [fcoeDiscoveredInfo cget -
sourceId]
    set firstPgid[string range $pgidStringFcoe 7 8]

    if {[interfaceTable getFirstInterface
interfaceTypeNpiv ]} {
        errorMsg "Error adding interfaceTypeNpiv
to interfaceTable on $chassisId $cardId $portId."
        set retCode $::TCL_ERROR
    }
}

```

```

        set interfaceDescriptionNpiv [interfaceEntry cget
-description]

        if {[interfaceTable getFcoeDiscoveredInfo
$interfaceDescriptionNpiv]} {
            errorMsg "Error getting Fcoe Discovered
table for $interfaceDescriptionNpiv on $chassisId $cardId $portId."
            set retCode $::TCL_ERROR
        }
        set sourceIdArray($chassisId,$cardId,$portId,npiv)
[fcoeDiscoveredInfo cget -sourceId]
ixPuts ">>>> $interfaceDescription
DiscoveredInfo [fcoeDiscoveredInfo cget -sourceId]"

        set pgidStringNpiv      [fcoeDiscoveredInfo cget -
sourceId]
        set secondPgid[string range $pgidStringNpiv 7 8]

    }

if {[configureFcoeStreams $portList sourceIdArray]} {
    errorMsg "Error configureFcoeStreams"
    set retCode $::TCL_ERROR
}

if {$retCode == $::TCL_OK } {

    set txFrames 2000;# 1000 FCoE and 1000 Npiv

    ixClearTimeStamp portList
    ixClearStats portList
    ixStartCapture portList
    ixStartPacketGroups portList
    ixStartTransmit portList
    after 2000

    ixStopTransmit portList
    ixCheckTransmitDone portList
    ixStopPacketGroups portList
    ixStopCapture portList

    ixRequestStats portList
    set expectedNumGroups 1455

    foreach port $portList {
        scan $port "%d %d %d" chassisId cardId
portId

        if {[statList get $chassisId $cardId
$portId]} {
            errorMsg "Error getting stats for
$chassisId $cardId $portId."
            set retCode $::TCL_ERROR
        }
        ixPuts " fcoeRxSharedStat1 [statList cget
-fcoeRxSharedStat1]"
        ixPuts " fcoeRxSharedStat2 [statList cget
-fcoeRxSharedStat2]"

        # Now get the statistics back
        # First a get for all of the packet groups
        if [packetGroupStats get $chassisId $cardId
$portId 0 $expectedNumGroups ] {

```

```
                errorMessage "Error in
packetGroupStats get on $chassisId $cardId $portId"
                                set retCode $::TCL_ERROR
                }
                ixPuts " numGroups [packetGroupStats cget
-numGroups]"
        }
}

# shut down FCoE Server
set pcpuFcoeSrvStop {/bin/ps& -ef | /bin/grep 'fcoeserver' | /
bin/grep -v grep | /shared/chassis/arch/usr/bin/awk '{print $1}' |
/usr/bin/xargs kill -9}
if {[issuePcpuCommand portList $pcpuFcoeSrvStop]} {
    errorMessage "Failed to kill FCoE server on $chassisId $cardId
$portId"
    set retCode $::TCL_ERROR
} else {
    errorMessage "FCoE Server Stopped..."
}

package forget IxTclServices
return $retCode
}

# Run the test
fcoeMainTest $portList
```

SEE ALSO

fcoeDiscoveredInfo, fcoeProperties, fibreChannel

NAME - fcoeDiscoveredInfo

fcoeDiscoveredInfo - configure FCoE discovery function.

SYNOPSIS	fcoeDiscoveredInfo <i>sub-command options</i>
DESCRIPTION	FCoE ports discover other ports within a communication path.
STANDARD OPTIONS	
destinationIdList	<i>Read only</i> . List of destination IDs.
discoveredVlanIds	<i>Read only</i> . The list of IDs discovered from the VLAN Discovery notification.
fabricAssignedMacAddress	<i>Read only</i> . (Only if FIP is enabled) The MAC address assigned by the Fabric. (<i>default</i> = '00 00 00 00 00 00')
fabricFcMap	<i>Read only</i> . (Only if FIP is enabled) Obtained from the Discovery Advertisement. (<i>default</i> = '0E.FC.00')
fabricMacAddress	<i>Read only</i> . MAC address of the Fabric (<i>default</i> = '00 00 00 00 00 00')
fabricName	<i>Read only</i> . (Only if FIP is enabled) The Fabric name obtained from the Discovery Advertisement. (<i>default</i> = 00:00:00:00:00:00)
priority	<i>Read only</i> . (Only if FIP is enabled) The priority of the Fabric we are logged into. (<i>default</i> = 128)
sourceId	<i>Read only</i> . Source ID assigned by the Fabric (<i>default</i> = '00.00.00')
status	<i>Read only</i> . Textual description of the status of the interface (<i>default</i> = 0)

Option	Value	Usage
fcoeStatusUnknown	0	(<i>default</i>) FCoE status unknown
fcoeStatusFLogiComplete	1	FCoE status Fabric Login complete
fcoeStatusFLogiFailed	2	FCoE status Fabric Login failed
fcoeStatusPLogiComplete	3	FCoE status Port Login complete
fcoeStatusPLogiFailed	4	FCoE status Port Login failed
fcoeStatusScrComplete	5	FCoE status Scr complete
fcoeStatusScrFailed	6	FCoE status Scr failed
fcoeStatusFDiscComplete	7	FCoE status FCoE Discovered complete
fcoeStatusFDiscFailed	8	FCoE status FCoE Discovered failed
fcoeStatusNsRegistrationComplete	9	FCoE status FCoE NS registration complete
fcoeStatusNsRegistrationFailed	10	FCoE status FCoE NS registration failed
fcoeStatusDiscoverySolicitationComplete	11	FCoE status Discovery Solicitation complete
fcoeStatusDiscoverySolicitationFailed	12	FCoE status Discovery Solicitation failed

Option	Value	Usage
fcoeStatusVlanDiscoveryComplete	13	FCoE status VLAN Discovery complete
fcoeStatusVlanDiscoveryFailed	14	FCoE status VLAN Discovery failed
fcoeStatusPRLIComplete	15	PRLI status complete
fcoeStatusPRLIFailed	16	PRLI status failed
fcoeStatusNSQueryComplete	17	NS Query status complete
fcoeStatusNSQueryFailed	18	NS Query status failed
fcoeStatusPLogiSent	19	PLOGI status failed

switchName

Read only. (Only if FIP is enabled) The switch name obtained from the Discovery Advertisement. (*default = 00:00:00:00:00:00:00:00*)

COMMANDS

The *fcoeDiscoveredInfo* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

fcoeDiscoveredInfo cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **fcoeDiscoveredInfo** command.

fcoeDiscoveredInfo config option value

Modify the fcoeDiscoveredInfo configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

fcoeDiscoveredInfo setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under [fcoe](#).

SEE ALSO

[fcoe](#), [fcoeProperties](#), [fibreChannel](#)

NAME - fcoeNameServer

fcoeNameServer- configure FCoE Name Server

SYNOPSIS	fcoeNameServer <i>sub-command options</i>
DESCRIPTION	The <i>fcoeNameServer</i> command is used to configure the FCoE Name Server.
<hr/>	
STANDARD OPTIONS	
enableRnnId <i>true/false</i>	Register Node Name (RNN_ID) (<i>default = true</i>)
enableRcsId <i>true/false</i>	Register Class of Service (RCS_ID) (<i>default = false</i>)
enableRftId <i>true/false</i>	Register RC-4 Types (RFT_ID) (<i>default = true</i>)
enableRpnId <i>true/false</i>	Register Port Name (RPN_ID) (<i>default = false</i>)
enableRptId <i>true/false</i>	Register Port Type (RPT_ID) (<i>default = false</i>)
enableRspnId <i>true/false</i>	Register Symbolic Port Name (RSPN_ID) (<i>default = false</i>)
enableRsnnNn <i>true/false</i>	Register Symbolic Node Name (RSNN_NN) (<i>default = false</i>)
symbolicPortName	A user-defined string to identify a port, for example 'Ixia Port 1'.
symbolicNodeName	A user-defined string to identify a node, for example 'Ixia Node 1'.
<hr/>	
COMMANDS	The <i>fcoeNameServer</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

fcoeNameServer **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **fcoeNameServer** command.

fcoeNameServer **config** *option value*

Modify the fcoeNameServer configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

fcoeNameServer **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *fcoe*.

SEE ALSO

fcoe, fcoeDiscoveredInfo, fibreChannel

NAME - fcoePlogi

fcoePlogi configures the FCoE Name Server plogi designation

SYNOPSIS	fcoePlogi <i>sub-command options</i>
-----------------	--------------------------------------

DESCRIPTION	The <i>fcoePlogi</i> command is used to configure the FCoE Name Server plogi designation.
--------------------	---

STANDARD OPTIONS

enable true/false	Enables Port login (plogi) to specified Destination ID (<i>default = true</i>)
------------------------------	--

destinationMode	Specifies destination mode. (<i>default = fcoeFabricProvidedMacAddress</i>)
------------------------	---

Option	Value
plogiDestinationId	0
plogiWwpn	1

destinationId	Destination identifier (<i>default = '00.00.00'</i>)
----------------------	--

wwpn	Source port Worldwide Name - a Name_identifier that is worldwide unique, represented by a 64-bit value (<i>default = '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'</i>)
-------------	---

COMMANDS	The <i>fcoePlogi</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

fcoePlogi **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **fcoePlogi** command.

fcoePlogi **config** *option*

Modify the **fcoePlogi** configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

fcoePlogi **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES	See example under <i>fcoe</i> .
-----------------	---------------------------------

SEE ALSO	<i>fcoe, fcoediscoveredInfo, fibreChannel</i>
-----------------	---

NAME - fcoeProperties

fcoeProperties- configure properties for FCoE

SYNOPSIS

fcoeProperties *sub-command options*

DESCRIPTION

Fibre Channel over Ethernet (FCoE) is a method of communicating data for streams and protocols.

STANDARD OPTIONS

addressingMode

Specifies the FCoE addressing mode to be used by the n-port. (default = *fcoeFabricProvidedMacAddress*)

Option	Value
fcoeFabricProvidedMacAddress	0
fcoeServerProvidedMacAddress	1
fcoeBoth	2

bufferToBufferRxSize

Maximum buffer-to-buffer Receive_Data_Field specified by the Fabric (*default = 2112*)

destinationId

Use to configure the Fibre Channel Frame header destination ID. (*default = 01.b6.69*)

enableAutoPlogi

Automatically enables PLOGI to all the ports that are advertised by the fabric, or to PLOGI to a subset of the variable ports that belong to a specified domain. (*default = false*)

enableFip

true/false

Enables FIP (FCoE Initialization Protocol) If enabled, the interface uses FIP for its initialization. Otherwise, it uses Cisco adhoc standard. (*default = true*)

enableNs

true/false

Enables registration to Name Server (*default = false*)

enableNSQuery

If true, enables Name Server Query parameters for this FCoE server.

enablePlogi

true/false

Enables Port login to specified Destination ID (*default = false*)

enablePRLI

If true, enables Process Login parameters. The PRLI request is used to establish the operating environment between a group of related processes at the originating Nx_Port and a group of related processes at the responding Nx_Port. If true, this option causes the state machine to attempt a process login.

enableResetFip

Discovery

If true, resets FIP Discovery tag.

enableSCR

true/false

If set to true, the ENode registers for any changes with the Fabric by sending a State Change Registration packet. (*default = false*)

enableVlanDiscovery	Enables VLAN Discovery (<i>default = false</i>)
enableUntaggedVlanDiscovery	Enables untagged VLAN Discovery (<i>default = true</i>)
enableVnPortKeepAlive	If true, VN port sends periodic keep alives.
s	
enableENodeKeepAlive	If true, ENode sends periodic keep alives.
s	
maxSize	Enter the maximum FCoE size (<i>default = 2158</i>)
resetFipDiscoveryRetries	If set to true, retries FIP discovery for the selected number of times.
scrOption	If <i>enableSCR</i> is set to true, <i>scrOption</i> becomes true. The registration function options for SCR are Fabric Detected, Nx Port Detected, Full Registration.
sourceNodeWWN	Source node Worldwide Name - a Name_identifier that is worldwide unique, represented by an 8-byte hex value. (<i>default = '00 ... 00'</i>)
sourceOui	Use to configure the source Organization Unique Identifier. (<i>default = 0e.fc.00</i>)
sourcePortWWN	Source port Worldwide Name - a Name_identifier that is worldwide unique, represented by an 8-byte hex value. (<i>default = '00 ... 00'</i>)
vendorId	Enter a string to be used in vendor-specific messages. (<i>default = '00 00 00 00 00 00 00 00'</i>)

COMMANDS

The *fcoeProperties* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

fcoeProperties cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **fcoeProperties** command.

fcoeProperties config *option value*

Modify the fcoeProperties configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

fcoeProperties setDefault

Sets to IxTclHal default values for all configuration options.

fcoeProperties addTlv

Adds a TLV to fcoeProperties. The values are available in the [*fipTlv*](#) command.

fcoeProperties **delTlv** *tlvIndex*

Deletes the TLV associated with this FCoE property set at the specified index. The index of the first entry is 1. The values are available in the *fipTlv* command. Specific errors are:

- The indexed entry does not exist in the list.
- Invalid index.

fcoeProperties **getTlv** *tlvIndex*

Retrieves the TLV associated with this FCoE property set at the specified index. The index of the first entry is 1. The values are available in the *fipTlv* command. Specific errors are:

- The indexed entry does not exist in the list.

fcoeProperties **getFirstTlv**

Retrieves the first TLV associated with this FCoE property set. The values are available in the *fipTlv* command. Specific errors are:

- There are no entries in the list.

fcoeProperties **getNextTlv**

Retrieves the next TLV associated with this FCoE property set. The values are available in the *fipTlv* command. Specific errors are:

- There are no more entries in the list.

fcoeProperties **removeAllTlvs**

Deletes all of the TLVs associated with this FCoE property set.

fcoeProperties **addPlogi**

Adds a PLOGI to fcoeProperties. The values are available in the *fcoePlogi* command.

fcoeProperties **delPlogi** *plogiIndex*

Deletes the PLOGI associated with this FCoE property set at the specified index. The index of the first entry is 1. The values are available in the *fcoePlogi* command. Specific errors are:

- The indexed entry does not exist in the list.
- Invalid index.

fcoeProperties **getPlogi** *plogiIndex*

Retrieves the PLOGI associated with this FCoE property set at the specified index. The index of the first entry is 1. The values are available in the *fcoePlogi* command. Specific errors are:

- The indexed entry does not exist in the list.

fcoeProperties **getFirstPlogi**

Retrieves the first PLOGI associated with this FCoE property set. The values are available in the *fcoePlogi* command. Specific errors are:

- There are no entries in the list.

fcoeProperties getNextPlogi

Retrieves the next PLOGI associated with this FCoE property set. The values are available in the *fcoePlogi* command. Specific errors are:

- There are no more entries in the list.

fcoeProperties removeAllPlogis

Deletes all of the PLOGIs associated with this FCoE property set.

EXAMPLES

```
package req IxTclHal

set hostnameLoopback

if {[ixConnectToChassis $hostname]} {
    errorMsg "error connecting $hostname chassis"
    return "FAIL"
}

set chassisId      [chassis cget -id]
set cardId         1
set portId         1
set streamId       1
set portList [list [list $chassisId $cardId $portId] ]

if {![port isValidFeature $chassisId $cardId $portId $::portFeatureDataCenterMode]} {
    errorMsg "portFeatureDataCenterMode is not valid on $chassisId $cardId $portId"
    return "FAIL"
}

port setFactoryDefaults $chassisId $cardId $portId
port config -enableDataCenterMode$::true

if {[port set $chassisId $cardId $portId]} {
    errorMsg "Error setting port on $chassisId $cardId $portId"
}

# Configure FIP interfaces

if {[interfaceTable select $chassisId $cardId $portId]} {
    errorMsg "Error selecting interfaceTable on $chassisId $cardId $portId."
}

fcoeProperties setDefault
fcoeProperties removeAllTlvs
fcoeProperties removeAllPlogis

fcoeProperties config -sourcePortWWN           "10 00 00 00 05 F9 A9 B0"
fcoeProperties config -sourceNodeWWN          "20 00 00 00 05 F9 A9 B0"
fcoeProperties config -destinationId        "01.b6.69"
fcoeProperties config -sourceOui            "0e.fc.00"
fcoeProperties config -bufferToBufferRxSize   2112
fcoeProperties config -enableNs              false
fcoeProperties config -enablePlogi           false
fcoeProperties config -enableFip             true
fcoeProperties config -maxSize              2158
fcoeProperties config -addressingMode       0
fcoeProperties config -vendorId            "00 00 00 00 00 00 00 00"
fcoeProperties config -enableVlanDiscovery  false
```

```

fcoeNameServer setDefault

fcoeNameServer config -enableRnnId           false
fcoeNameServer config -symbolicPortName      Ixial
fcoeNameServer config -symbolicNodeName       Ixia2

fipTlv          setDefault
fipTlv config -type   1
fipTlv config -value  "11 22 33 44"

if {[fcoeProperties addTlv]} {
    Trace "Error adding tlv to fcoeProperties on $chassisId $cardId $portId."
    set retCode "FAIL"
}

fcoePlogi setDefault
fcoePlogi config -enable 1
fcoePlogi config -destinationMode 1
fcoePlogi config -wwpn "00 11 00 11 33 11 11 11"

if {[fcoeProperties addPlogi]} {
    Trace "Error adding tlv to fcoeProperties on $chassisId $cardId $portId."
    set retCode "FAIL"
}

fcoePlogi setDefault
fcoePlogi config -enable 1
fcoePlogi config -destinationMode 0
fcoePlogi config -destinationId "11 22 33"

if {[fcoeProperties addPlogi]} {
    Trace "Error adding tlv to fcoeProperties on $chassisId $cardId $portId."
    set retCode "FAIL"
}

set interfaceDescription"FipInterface"

interfaceEntry setDefault

interfaceEntry config -enable      true
interfaceEntry config -enableFlogi true
interfaceEntry config -description $interfaceDescription
interfaceEntry config -macAddress {00 00 2D 5C C1 3C}
if {[interfaceTable addInterface interfaceTypeConnected]} {
    errorMsg "Error adding interfaceTypeConnected to interfaceTable on $chassisId $cardId
$portId."
}

#### Interface entry type - interfaceTypeNpiv
interfaceEntry clearAllItems addressTypeIPv6
interfaceEntry clearAllItems addressTypeIPv4
interfaceEntry setDefault

npivProperties setDefault
npivProperties removeAllPlogis
npivProperties config -sourcePortWWN           "10 00 1A FF FE 5E 3B 36"
npivProperties config -sourceNodeWWN          "20 00 1A 00 00 5E 3B 36"
npivProperties config -destinationId         "01.b6.66"
npivProperties config -bufferToBufferRxSize  2112
npivProperties config -enableNs              true
npivProperties config -enablePlogi           false

```

```

fcoePlogi setDefault
fcoePlogi config -enable 1
fcoePlogi config -destinationMode 1
fcoePlogi config -wwpn "11 11 00 00 88 88 88 88"

if {[npivProperties addPlogi]} {
    Trace "Error adding tlv to fcoeProperty"
    Trace "Error adding tlv to fcoeProperties on $chassId $cardId $portId."
    set retCode "FAIL"
}

fcoePlogi setDefault
fcoePlogi config -enable 1
fcoePlogi config -destinationMode 0
fcoePlogi config -destinationId "11 22 33"

if {[npivProperties addPlogi]} {
    Trace "Error adding tlv to fcoeProperty"
    Trace "Error adding tlv to fcoeProperties on $chassId $cardId $portId."
    set retCode "FAIL"
}

interfaceEntry config -enable true
interfaceEntry config -description {InterfaceNPIV}
interfaceEntry config -connectedVia {FipInterface}

if {[interfaceTable addInterface interfaceTypeNpiv]} {
    errorMsg "Error calling interfaceTable addInterface interfaceTypeNpiv"
    set retCode $::TCL_ERROR
}

ixWriteConfigToHardware portList

# Verify FIP discovered information

if {[interfaceTable select $chassId $cardId $portId]} {
    errorMsg "Error selecting interfaceTable on $chassId $cardId $portId."
}

interfaceEntry setDefault
fcoeProperties setDefault

if {[interfaceTable getFirstInterface interfaceTypeConnected]} {
    errorMsg "Error adding interfaceTypeConnected to interfaceTable on $chassId $cardId $portId."
}

if {[interfaceTable requestDiscoveredTable]} {
    errorMsg "Error interfaceTable requestDiscoveredTable on $chassId $cardId $portId."
}

fcoeDiscoveredInfo setDefault
if {[interfaceTable getFcoeDiscoveredInfo $interfaceDescription]} {
    errorMsg "Error getting Fcoe Discovered table for $interfaceDescription on $chassId $cardId $portId."
}

ixPuts "fcoeDiscoveredInfo sourceId : [fcoeDiscoveredInfo cget -sourceId]"
ixPuts "fcoeDiscoveredInfo priority : [fcoeDiscoveredInfo cget -priority]"
ixPuts "fcoeDiscoveredInfo fabricAssgndMacAdr[fcoeDiscoveredInfo cget -fabricAssignedMacAddress]"
ixPuts "fcoeDiscoveredInfo switchName : [fcoeDiscoveredInfo cget -switchName]"
ixPuts "fcoeDiscoveredInfo fabricName : [fcoeDiscoveredInfo cget -fabricName]"

```

```
ixPuts "fcoeDiscoveredInfo fabricFcMap : [fcoeDiscoveredInfo cget -fabricFcMap]"
ixPuts "fcoeDiscoveredInfo discoveredVlanIds : [fcoeDiscoveredInfo cget -
discoveredVlanIds]"
ixPuts "fcoeDiscoveredInfo destinationIdList : [fcoeDiscoveredInfo cget -
destinationIdList]"
```

SEE ALSO

fcoe, fcoeDiscoveredInfo, fibreChannel, fcoePlogi, fipTlv

NAME - fcPlogi

fcPlogi-configures the FC Name Server plogi designation

SYNOPSIS	<i>fcPlogi sub-command options</i>
-----------------	------------------------------------

DESCRIPTION	The <i>fcPlogi</i> command is used to configure the FC Name Server plogi designation.
--------------------	---

STANDARD OPTIONS

enable <i>true/false</i>	Enables Port login (plogi) to specified Destination ID (<i>default = true</i>)
------------------------------------	--

destinationMode	Specifies destination mode.
------------------------	-----------------------------

Option	Value
plogiDestinationId	0
plogiWwpn	1

destinationId	Destination identifier (<i>default = '00.00.00'</i>)
----------------------	--

wwpn	Source port Worldwide Name - a Name_identifier that is worldwide unique, represented by a 64-bit value (<i>default = '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'</i>)
-------------	---

COMMANDS	The <i>fcPlogi</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

fcPlogi **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **fcPlogi** command.

fcPlogi **config** *option*

Modify the **fcPlogi** configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

fcPlogi **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES	See example under <i>fibreChannel</i> .
-----------------	---

SEE ALSO	<i>fibreChannel</i>
-----------------	---------------------

NAME - fcPort

fcPort-configures port for Fibre Channel

SYNOPSIS	<i>fcPort sub-command options</i>
DESCRIPTION	Fibre Channel (FC) port enables communicating data for streams and protocols.

STANDARD OPTIONS									
forceErrorMode	Allows to set error messages for frames of data. The types are as follows:								
	<table border="1"> <thead> <tr> <th>Option</th> <th>Usage</th> </tr> </thead> <tbody> <tr> <td><i>noErrors</i></td> <td>If true, does not send error messages.</td> </tr> <tr> <td><i>dontSendRRDY</i></td> <td>If true, does not send Receiver_Ready (R_RDY) Primitive error signal.</td> </tr> <tr> <td><i>dontSendRRDYEveryNFrames</i></td> <td>If true, does not send Receiver_Ready (R_RDY) Primitive error signal for each data frame.</td> </tr> </tbody> </table>	Option	Usage	<i>noErrors</i>	If true, does not send error messages.	<i>dontSendRRDY</i>	If true, does not send Receiver_Ready (R_RDY) Primitive error signal.	<i>dontSendRRDYEveryNFrames</i>	If true, does not send Receiver_Ready (R_RDY) Primitive error signal for each data frame.
Option	Usage								
<i>noErrors</i>	If true, does not send error messages.								
<i>dontSendRRDY</i>	If true, does not send Receiver_Ready (R_RDY) Primitive error signal.								
<i>dontSendRRDYEveryNFrames</i>	If true, does not send Receiver_Ready (R_RDY) Primitive error signal for each data frame.								

RRDYResponseDelay Mode	Allows to set response delays for R_RDY. The Receiver_Ready (R_RDY) Primitive Signal is used in the buffer-to-buffer Credit management mechanisms Validity of the frame is not implied by R_RDY. The types are as follows:										
	<table border="1"> <thead> <tr> <th>Option</th> <th>Usage</th> </tr> </thead> <tbody> <tr> <td><i>noDelay</i></td> <td>If true, does not set any delay for R_RDY response.</td> </tr> <tr> <td><i>fixedDelay</i></td> <td>If true, sets fixed delays in milliseconds for R_RDY response.</td> </tr> <tr> <td><i>randomDelay</i></td> <td>If true, sets random delays for R_RDY response.</td> </tr> <tr> <td><i>creditStarvation</i></td> <td>If true, programs a counter with delay value specified in the Hold R_RDY field. The counter starts counting down after it receives first frame. The port holds R_RDY for all frames received until counter reaches to 0. After counter reaches to 0, port sends out all accumulated R_RDY.</td> </tr> </tbody> </table>	Option	Usage	<i>noDelay</i>	If true, does not set any delay for R_RDY response.	<i>fixedDelay</i>	If true, sets fixed delays in milliseconds for R_RDY response.	<i>randomDelay</i>	If true, sets random delays for R_RDY response.	<i>creditStarvation</i>	If true, programs a counter with delay value specified in the Hold R_RDY field. The counter starts counting down after it receives first frame. The port holds R_RDY for all frames received until counter reaches to 0. After counter reaches to 0, port sends out all accumulated R_RDY.
Option	Usage										
<i>noDelay</i>	If true, does not set any delay for R_RDY response.										
<i>fixedDelay</i>	If true, sets fixed delays in milliseconds for R_RDY response.										
<i>randomDelay</i>	If true, sets random delays for R_RDY response.										
<i>creditStarvation</i>	If true, programs a counter with delay value specified in the Hold R_RDY field. The counter starts counting down after it receives first frame. The port holds R_RDY for all frames received until counter reaches to 0. After counter reaches to 0, port sends out all accumulated R_RDY.										

TOVMode	Allows to set Timeout Values. The types are as follows:				
	<table border="1"> <thead> <tr> <th>Option</th> <th>Usage</th> </tr> </thead> <tbody> <tr> <td><i>eDTOVMode</i></td> <td>Error_Detect_Timeout Value (E_D_TOV) is a short timeout value. The E_D_TOV is used as the timeout value for detecting an error condition.</td> </tr> </tbody> </table>	Option	Usage	<i>eDTOVMode</i>	Error_Detect_Timeout Value (E_D_TOV) is a short timeout value. The E_D_TOV is used as the timeout value for detecting an error condition.
Option	Usage				
<i>eDTOVMode</i>	Error_Detect_Timeout Value (E_D_TOV) is a short timeout value. The E_D_TOV is used as the timeout value for detecting an error condition.				

Option	Usage
<i>rATOVMode</i>	Resource_Allocation_Timeout Value (R_A_TOV) is a long timeout value. The R_A_TOV is used as the timeout value for determining when to Reinstate a Recovery_Qualifier.
<i>rTTTOVMode</i>	The Receiver_Transmitter timeout value (R_T_TOV) is used by the receiver logic to detect Loss-of-Synchronization. The default value for R_T_TOV is 100 milliseconds. A shorter value of 100 microseconds is also allowed.
<i>overrideTOVMode</i>	If true, error detection overrides 10,000 milliseconds.
<i>fromLoginMode</i>	If true, obtains response from login ID.

doNotSendRRDYAfterNFrames

If true, the transmitting port does not send R_RDY delays after *n* number of frames.

enableAutoNegotiate

Not used for FC.

enableTxIgnoreAvailableCredits

If true, the transmitting port does not listen to flow control. It keeps transmitting packets irrespective of available credits.

fixedDelayValue

If true, signifies fixed R_RDY response delays in μ s.

bbCredit

Buffer-to-buffer Credit is the number of received buffers supported by an FC Port for receiving Class 1 and 6/SOFc1, Class 2, or Class 3 frames. The minimum or default value of BB_Credit is one.

bbSCN

The buffer-to-buffer State Change Number. It is the log2 of BB_Credit Recovery modulus.

The default value is 0.

minDelayForRandom

If true, sets the minimum delay in milliseconds.

maxDelayForRandom

If true, sets the maximum delay in milliseconds.

COMMANDS

The *fcPort* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

fcPort setDefault

Sets to IxTclHal default values for all configuration options.

fcPort set

Sets the current configuration of the *fcPort* for the indicated port. Call this command before calling *fcPort get option value* to get the value of the configuration option.

fcPort get

Gets the current configuration of the *fcPort* for the indicated port.

EXAMPLES

See under [*fibreChannel*](#).

SEE ALSO

fibreChannel

NAME - fcProperties

fcProperties-configure properties for Fibre Channel

SYNOPSIS	fcProperties <i>sub-command options</i>
DESCRIPTION	Fibre Channel (FC) is a method of communicating data for streams and protocols.
STANDARD OPTIONS	
bufferToBufferRxSize	Maximum buffer-to-buffer Receive_Data_Field specified by the Fabric (<i>default = 2112</i>)
destinationId	Use to configure the Fibre Channel Frame header destination ID. (<i>default = 01.b6.69</i>)
enableAutoPlogi	Automatically enables PLOGI to all the ports that are advertised by the fabric, or to PLOGI to a subset of the variable ports that belong to a specified domain. (<i>default = false</i>)
enableNs <i>true/false</i>	Enables registration to Name Server (<i>default = false</i>)
enableNSQuery	If true, enables Name Server Query parameters for this FC server.
enablePlogi <i>true/false</i>	Enables Port login to specified Destination ID (<i>default = false</i>)
enablePRLI	If true, enables Process Login parameters. The PRLI request is used to establish the operating environment between a group of related processes at the originating Nx_Port and a group of related processes at the responding Nx_Port. If true, this option causes the state machine to attempt a process login.
enableSCR <i>true/false</i>	If set to true, the ENode registers for any changes with the Fabric by sending a State Change Registration packet. (<i>default = false</i>)
scrOption	If <i>enableSCR</i> is set to true, <i>scrOption</i> becomes true. The registration function options for SCR are Fabric Detected, Nx Port Detected, Full Registration.
sourceNodeWWN	Source node Worldwide Name - a Name_identifier that is worldwide unique, represented by an 8-byte hex value. (<i>default = '00 ... 00'</i>)
sourceOui	Use to configure the source Organization Unique Identifier. (<i>default = 0e.fc.00</i>)
sourcePortWWN	Source port Worldwide Name - a Name_identifier that is worldwide unique, represented by an 8-byte hex value. (<i>default = '00 ... 00'</i>)
COMMANDS	The <i>fcProperties</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

fcProperties addPlogi

Adds a PLOGI to fcProperties.

fcProperties delPlogi *plogiIndex*

Deletes the PLOGI associated with this FC property set at the specified index.
The index of the first entry is 1.

fcProperties getPlogi *plogiIndex*

Retrieves the PLOGI associated with this FC property set at the specified index.
The index of the first entry is 1.

fcProperties getFirstPlogi

Retrieves the first PLOGI associated with this FC property set. Specific errors are:

- There are no entries in the list.

fcProperties getNextPlogi

Retrieves the next PLOGI associated with this FC property set. Specific errors are:

- There are no more entries in the list.

fcProperties removeAllPlogis

Deletes all of the PLOGIs associated with this FC property set.

EXAMPLES

See under *fibreChannel*.

SEE ALSO

fibreChannel

NAME - fcSOF

fcSOF-inserts Fibre Channel Start-of-Frame (SOF) delimiter. It is an Ordered Set that immediately precedes the frame content.

SYNOPSIS	fcSOF <i>sub-command options</i>
-----------------	----------------------------------

DESCRIPTION	The Start-of-Frame (SOF) delimiter is an Ordered Set that immediately precedes the frame content.
--------------------	---

STANDARD OPTIONS

fcSOFDelimiter	The multiple SOF delimiters defined for Sequence control are as follows:
-----------------------	--

Option	Usage
<i>fcSOFc1</i>	The SOFc1 is used for all frames except the first frame of a Sequence for Class 1 service or Class 6 service.
<i>fcSOFi1</i>	The SOFi1 is used for all frames except the first frame of a Sequence for Class 1 service or Class 6 service.
<i>fcSOFn1</i>	The SOFn1 is used for all frames except the first frame of a Sequence for Class 1 service or Class 6 service.
<i>fcSOFi2</i>	The SOFi2 is used on the first frame of a Sequence for Class 2 service.
<i>fcSOFn2</i>	The SOFn2 is used for all frames except the first frame of a Sequence for Class 2 service.
<i>fcSOFi3</i>	The SOFi3 is used on the first frame of a Sequence for Class 3 service.
<i>fcSOFn3</i>	The SOFn3 is used for all frames except the first frame of a Sequence for Class 3 service.
<i>fcSOFc4</i>	The SOFc4 is used on the first frame of a Connect for Class 4 service.
<i>fcSOFi4</i>	The SOFi4 is used on the first frame of a Sequence for Class 4 service.
<i>fcSOFn4</i>	The SOFn4 is used for all frames except the first frame of a Sequence for Class 4 service.
<i>fcSOFF</i>	If an Nx_Port or Fx_Port receives a Class F frame, indicated by an SOFF delimiter, it is discarded by the Nx_Port or Fx_Port. The receiving Nx_Port or Fx_Port may send an R_RDY.

COMMANDS	The <i>fcSOF</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

fcSOF setDefault *option*

Sets to IxTclHal default values for all configuration options.

fcSOF set *option*

Sets the current configuration of the *fcSOF* for the indicated port. Call this command before calling *fcSOF get option value* to get the value of the configuration option.

fcSOF get *option*

Gets the current configuration of the *fcSOF* for the indicated port.

EXAMPLES

See under *fibreChannel*.

SEE ALSO

fibreChannel

NAME - fecError

fecError - insert FEC errors

SYNOPSIS

fecError *sub-command options*

DESCRIPTION

Forward Error Correction (FEC) is a method of communicating data that corrects errors in transmission on the receiving end. Prior to transmission, the data is put through a predetermined algorithm that adds extra bits specifically for error correction to any character or code block. If the transmission is received in error, the correction bits are used to check and repair the data. This feature is only available for certain port types; this may be tested through the use of the *portisValidFeature... portFeatureFec* command. FEC insertion must be enabled through the use of the *opticalDigitalWrapper* command.

The *fecError* command allows you to inject FEC errors into transmitted data. Three distinct modes are controlled by the *injectionMode* option:

- Single: a single instance of an error is inserted.
- Rate: errors are inserted at one of a set of pre-determined rates as controlled by the *errorRate* option.
- Burst: continuous bursts of errors is inserted as determined by the *subrow*, *burstSize*, *offset*, *errorBits* and *numberOfRowsToSkip* options.

Single errors are inserted with the *injectError* sub-command and the *start* and *stop* commands are used to start and stop rate and burst error insertion.

STANDARD OPTIONS

burstSize

The number of consecutive bytes (up to 15) after the 1st corrupted byte in each 255-byte sub-row to also be corrupted. Thus if Burst Size = 15, then 16 bytes are corrupted in each selected sub-row. Default Burst Size = 0, meaning only 1 byte is corrupted. Values between 8 and 15 results in uncorrectable errors. (*default = 0*)

errorBits

The OR'ing of a set of bits to be errored. The default value of 0x01 shows error the low order bit. The value of 0x81 errors both the high order and low order bits. (*default = 1*)

errorRate

One of a set of pre-defined error rates.

Option	Value	Usage
<i>fecRate_0996_e02_correctable</i>	0	(<i>default</i>) 0.996×10^{-2} correctable
<i>fecRate_1001_e03_correctable</i>	1	1.001×10^{-3} correctable
<i>fecRate_1001_e04_correctable</i>	2	1.001×10^{-4} correctable
<i>fecRate_1001_e05_correctable</i>	3	1.001×10^{-5} correctable
<i>fecRate_1000_e06_correctable</i>	4	1.000×10^{-6} correctable
<i>fecRate_1000_e07_correctable</i>	5	1.000×10^{-7} correctable
<i>fecRate_1000_e08_correctable</i>	6	1.000×10^{-8} correctable

Option	Value	Usage
<code>fecRate_1000_e09_correctable</code>	7	1.000×10^{-9} correctable
<code>fecRate_1000_e10_correctable</code>	8	1.001×10^{-10} correctable
<code>fecRate_1000_e11_correctable</code>	9	1.000×10^{-11} correctable
<code>fecRate_1000_e12_correctable</code>	10	1.0010×10^{-12} correctable
<code>fecRate_0960_e02_uncorrectable</code>	11	0.960×10^{-2} uncorrectable
<code>fecRate_1000_e03_uncorrectable</code>	12	1.000×10^{-3} uncorrectable
<code>fecRate_1000_e04_uncorrectable</code>	13	1.000×10^{-4} uncorrectable
<code>fecRate_1000_e05_uncorrectable</code>	14	1.000×10^{-5} uncorrectable
<code>fecRate_1000_e06_uncorrectable</code>	15	1.000×10^{-6} uncorrectable
<code>fecRate_1000_e07_uncorrectable</code>	16	1.000×10^{-7} uncorrectable
<code>fecRate_1000_e08_uncorrectable</code>	17	1.000×10^{-8} uncorrectable
<code>fecRate_1000_e09_uncorrectable</code>	18	1.000×10^{-9} uncorrectable
<code>fecRate_1000_e10_uncorrectable</code>	19	1.001×10^{-10} uncorrectable

injectionMode

The mode of error injection.

Option	Value	Usage
<code>fecSingleErrorInjection</code>	0	(default) Indicates that a single error is inserted. The error is an argument to the <code>insertError</code> sub-command. This option must be set to this value when <code>insertError</code> is used.
<code>fecErrorRateInjection</code>	1	Continuously inserts errors at one of a set of pre-determined rate indicated in the <code>errorRate</code> option.
<code>fecBurstErrorInjection</code>	2	Inserts continuous bursts of errors as indicated by the <code>subrow</code> , <code>burstSize</code> , <code>offset</code> , <code>errorBits</code> and <code>numberOfRowsToSkip</code> options.

numberOfRowsToSkip

The number of rows to skip between error insertion burst. (default = 0)

offset

The offset within the subrow to start injecting errors. Byte 0 is the OH byte. (default = 1)

subrow

An OR'ing of bit to indicate which sub-row(s) to corrupt out of the 16 interleaved sub-rows. Each bit position represents one sub-row. Thus if `subrow` = 0xFFFF, then all sub-rows have errors on them. The low-order bit represents the first sub-row. (default = 0)

COMMANDS

The `fecError` command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

fecError cget option

Returns the current value of the configuration option given by `option`. `Option` may have any of the values accepted by the `fecError` command.

fecError config option value

Modify the configuration options of the `fecError`. If no `option` is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for `fecError`.

fecError get *chasID cardID portID*

Gets the current configuration of the *fecError* for the indicated port. Call this command before calling *fecError cget option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Unsupported feature

fecError injectError *fecErrorType chasID cardID portID*

Inject a single instance of the error indicated by *fecError* into the indicated port's stream. The valid options for *fecError* are:

Option	Value	Usage
<i>fecOnesError</i>	0	(default) Causes 0 bits to be changed to 1 bits.
<i>fecZerosError</i>	1	Causes 1 bits to be changed to 0 bits.
<i>fecBalancedError</i>	2	With an equal probability, a ones or zeros errors is inserted.
<i>fecUncorrectableError</i>	3	Causes uncorrectable errors to be inserted.

Specific errors are:

- No connection to a chassis
- Unsupported feature
- The port is being used by another user
- The value of *injectionMode* is not *fecSingleErrorInjection*

fecError set *chasID cardID portID*

Sets the configuration of the *fecError* in IxHAL for the port indicated by reading the configuration option values set by the *fecError config option value* command. Specific errors are:

- No connection to a chassis
- Unsupported feature
- The port is being used by another user
- The configured parameters are not valid for this port

fecError setDefault

Sets to IxTclHal default values for all configuration options.

fecError start *chasID cardID portID*

Starts the FEC error insertion process if *injectionMode* is *fecErrorRateInjection* or *fecBurstErrorInjection*. The *stop* sub-command must be used to stop error insertion. Specific errors are:

- No connection to a chassis
- Invalid port
- The port is being used by another user
- Unsupported feature
- The value of *injectionMode* is not *fecErrorRateInjection* or *fecBurstErrorInjection*.

fecError stop *chasID cardID portID*

Stops the FEC error insertion process if *injectionMode* is *fecErrorRateInjection* or *fecBurstErrorInjection*. Specific errors are:

- No connection to a chassis

- Invalid port
- The port is being used by another user
- Unsupported feature
- The value of *injectionMode* is not *fecErrorRateInjection* or *fecBurstErrorInjection*.

EXAMPLES

```

package req IxTclHal
set host localhost
set username test

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chassis [ixGetChassisID $host]
set card 69
set port 1

# Useful port lists
set portList [list [list $chassis $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

port      setFactoryDefaults $chassis $card $port

# Make sure to enable the use of FEC
opticalDigitalWrapper config -enableFec true
opticalDigitalWrapper config -payloadType
optDigWrapperPayloadType03
if [opticalDigitalWrapper set $chassis $card $port] {
    ixPuts $::ixErrorInfo
    return 1
}

# Inject a single balanced error
fecError setDefault
fecError  config      -injectionMode fecSingleErrorInjection
if [fecError  set $chassis $card $port] {
    ixPuts $::ixErrorInfo
    return 1
}
ixWriteConfigToHardware      portList

```

```

if [fecError injectError fecBalancedError $chassis $card $port]
{
    ixPuts $::ixErrorInfo
    return 1
}

# Setup a continuous error rate
fecError setDefault
fecError config -injectionMode fecErrorRateInjection
fecError config -errorRate fecRate_1000_e06_correctable
if [fecError set $chassis $card $port] {
    ixPuts $::ixErrorInfo
    return 1
}
ixWriteConfigToHardware portList
if [fecError start $chassis $card $port] {
    ixPuts $::ixErrorInfo
    return 1
}
after 1000
if [fecError stop $chassis $card $port] {
    ixPuts $::ixErrorInfo
    return 1
}

# Setup a burst rate
fecError setDefault
fecError config -injectionMode fecBurstErrorInjection
fecError config -subrow 0x0020
fecError config -burstSize 4
fecError config -offset 1
fecError config -errorBits 3
fecError config -numberOfRowsToSkip 2
if [fecError set $chassis $card $port] {
    ixPuts $::ixErrorInfo
    return 1
}
ixWriteConfigToHardware portList
if [fecError start $chassis $card $port] {
    ixPuts $::ixErrorInfo
    return 1
}
after 1000
if [fecError stop $chassis $card $port] {
    ixPuts $::ixErrorInfo
    return 1
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*opticalDigitalWrapper*

NAME - fibreChannel

fibreChannel-supports FC header and trailer in streams

SYNOPSIS

fibreChannel *sub-command options*

DESCRIPTION

The *fibreChannel* command supports FC header and trailer in streams.

STANDARD OPTIONS

abortSequenceCondition

Use to configure the Fibre Channel frame control abort sequence condition. (*default = 0*)

When Sequence Context is Initiator (0), available options are:

Option	Value	Usage
fibreChannelContinue	0	(default) Continue sequence
fibreChannelPerformAbts	1	Abort sequence perform ABTS
fibreChannelStop	2	Stop sequence
fibreChannelRetransmissionRequested	3	Immediate sequence retransmission requested

When Sequence Context is Recipient (1) available options are:

Option	Value	Usage
fibreChannelDiscardMultiple	0	(default) Abort discard multiple sequences
fibreChannelDiscardSingle	1	Abort discard a single sequence
fibreChannelProcessPolicy	2	Process policy with infinite buffers
fibreChannelDiscardMultipleWithRetransmission	3	Discard multiple sequences with immediate retransmission

ackForm

Use to configure the Fibre Channel frame control ack form bits. (*default = 0*)

Available options are:

Option	Value	Usage
fibreChannelNoAssistanceProvided	0	(default) No assistance provided
fibreChannelAck1Required	1	Ack1 required
fibreChannelReserved	2	Reserved
fibreChannelAck0Required	3	Ack0 required

continueSequenceCondition

Use to configure the Fibre Channel frame control continue sequence condition. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelNoInformation	0	(default) No information
fibreChannelFollowImmediately	1	Sequence to follow immediately
fibreChannelFollowSoon	2	Sequence to follow soon

Option	Value	Usage
fibreChannelFollowDelayed	3	Sequence to follow delayed

csControlOrPriority

Use to configure the Fibre Channel frame control class specific control or priority bit. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelCsCtl	0	(default) Cs Ctl
fibreChannelPriority	1	Priority

csControlOrPriority Value

Use to configure the Fibre Channel CS control or priority value which depends on the control/priority bit set in the frame control. See *enableUseFcControlBits*. (*default = 0x00*)

dataFieldControl

Use to configure the Fibre Channel Frame header data field. (*default = 0x00*)

destinationId

Use to configure the Fibre Channel Frame header destination ID. (*default = D1.D2.D3*)

enableBadFibreChannel Crc true/false

Use to enable the bad Fibre Channel checksum. (*default = true*)

enableUseFcControlBits true/false

Use to enable the Frame Control bit by bit configuration. (*default = false*)

endConnection

Use to configure the Fibre Channel frame control end connection bit. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelConnectionAlive	0	(default) Connection alive
fibreChannelConnectionPending	1	End of connection pending

endSequence

Use to configure the Fibre Channel frame control end sequence bit. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelEndSequenceOther	0	(default) End sequence other
fibreChannelEndSequenceLast	1	End sequence last

exchangeContext

Use to configure the Fibre Channel frame control exchange context bit. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelOriginator	0	(default) Exchange context originator
fibreChannelResponder	1	Exchange context responder

exchangeReassembly

Use to configure the Fibre Channel frame control exchange reassembly. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelExchangeReassembl yOff	0	(default) Exchange reassembly Off

Option	Value	Usage
fibreChannelExchangeReassemblyOn	1	Exchange reassembly On

fillBytes

Use to configure the Fibre Channel frame control fill bytes. (*default = 0*)

Available options are:

Option	Value	Usage
fibreChannelZeroHexByteFill	0	(default) 0 bytes of fill
fibreChannelOneHexByteFill	1	1 byte of fill
fibreChannelTwoHexByteFill	2	2 bytes of fill
fibreChannelThreeHexByteFill	3	3 bytes of fill

firstSequence

Use to configure the Fibre Channel frame control first sequence bit. (*default = 0*)

Available options are:

Option	Value	Usage
fibreChannelFirstSequenceOther	0	(default) First sequence other
fibreChannelFirstSequenceFirst	1	First sequence first

frameControl

Use to configure the Fibre Channel Frame header control bytes. If *enableUseFcControlBits* is set true, then this configuration is replaced by the bit by bit configuration. (*default = 00 00 00*)

lastSequence

Use to configure the Fibre Channel frame control last sequence bit. (*default = 0*)

Available options are:

Option	Value	Usage
fibreChannelLastSequenceOther	0	(default) Last sequence other
fibreChannelLastSequenceLast	1	Last sequence last

originatorExchangeCounter

Use to configure the Fibre Channel Frame header originator exchange ID counter. (*default = 0*)

Option	Value	Usage
fibreChannelIdle	0	(default) Idle
fibreChannelIncrement	1	Increment
fibreChannelDecrement	2	Decrement
fibreChannelContIncr	3	Continuous increment
fibreChannelContDecr	4	Continuous decrement
fibreChannelRandom	5	Random

originatorExchangeId

Use to configure the Fibre Channel Frame header originator exchange ID. (*default = '00 00'*)

parameter

Use to configure the Fibre Channel parameter. (*default = '00 00 00 00'*)

relativeOffsetPresent

Use to configure the Fibre Channel frame control relative offset present. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelRelativeOffsetDefined	0	(default) Parameter field defined
fibreChannelRelativeOffsetPresent	1	Relative offset present

responderExchangeId

Use to configure the Fibre Channel Frame header responder exchange ID. (*default = '00 00'*)

retransmittedSequence

Use to configure the Fibre Channel frame control retransmitted sequence bit. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelOriginal	0	(default) Original
fibreChannelRetransmission	1	Retransmission

routingControlInformation

Use to configure the Fibre Channel Frame header routing control information. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelUncategorizedInformation	0	(default) Uncategorized information
fibreChannelSolicitedData	1	Solicited data
fibreChannelUnsolicitedControl	2	Unsolicited control
fibreChannelSolicitedControl	3	Solicited control
fibreChannelUnsolicitedData	4	Unsolicited data
fibreChannelDataDescriptor	5	Data descriptor
fibreChannelUnsolicitedCommand	6	Unsolicited command
fibreChannelCommandStatus	7	Command status

routingControlType

Use to configure the Fibre Channel Frame header routing control type. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelDeviceDataFrames	0	(default) Device data frame
fibreChannelExtendedLinkServices	2	Extended link services
fibreChannelFc4LinkData	3	FC4 link data
fibreChannelVideoData	4	Video data
fibreChannelExtenderHeaders	5	Extended headers
fibreChannelBasicLinkServices	8	Basic link services
fibreChannelLinkControlFrame	12	Link control frame
fibreChannelExtendedRouting	15	Extended routing

sequenceContext

Use to configure the Fibre Channel frame control sequence context bit. (*default = 0*) Available options are:

Option	Value	Usage
fibreChannelInitiator	0	(default) Sequence context initiator

Option	Value	Usage
fibreChannelRecipient	1	Sequence context recipient

sequenceCount

Use to configure the Fibre Channel Frame header sequence count. (*default* = 0)

sequenceId

Use to configure the Fibre Channel Frame header sequence ID. (*default* = 0x00)

sequenceInitiative

Use to configure the Fibre Channel frame control sequence initiative bit. (*default* = 0) Available options are:

Option	Value	Usage
fibreChannelInitiativeHold	0	(default) Sequence initiative hold
fibreChannelInitiativeTransfer	1	Sequence initiative transfer

sourceId

Use to configure the Fibre Channel Frame header source ID. (*default* = '8D.8E.8F')

type

Use to configure the Fibre Channel Frame header type. (*default* = 0x00)

unidirectionalTransmit

Use to configure the Fibre Channel frame control unidirectional transmit bit. (*default* = 0) Available options are:

Option	Value	Usage
fibreChannelBidirectional	0	(default) Bidirectional
fibreChannelUnidirectional	1	Unidirectional

COMMANDS

The **fibreChannel** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

fibreChannel cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **fibreChannel** command.

fibreChannel config option value

Modify the configuration options of the filter. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **fibreChannel**.

fibreChannel decode capFrame chasID cardID portID

Decodes the FCoE packet and refreshes the IxTclHal object. Specific errors are:

- No connection to a chassis
- Invalid port
- The captured frame is not a valid fibreChannel packet

fibreChannel get chasID cardID portID

Gets the current FCoE settings from IxHal and refreshes IxTclHal object. Specific errors are:

- No connection to a chassis
- Invalid port number

fibreChannel setDefault

Sets to IxTclHal local default values for all configuration options.

fibreChannel set *chassisID cardID portID*

Sets the current FCoE settings from IxTclHal to local IxHAL. Specific errors are:

- No connection to a chassis
- Invalid port number
- Invalid feature
- The port is being used by another user
- The configured parameters are not valid for this port

EXAMPLES

```
package req IxTclHal

# Command Option Mode - Full (generate full configuration)

if {[isUNIX]} {
    if {[ixConnectToTclServer loopback]} {
        errorMsg "Error connecting to Tcl Server loopback"
    }
    return $::TCL_ERROR
}
#### Chassis list - {loopback} #####
ixConnectToChassis {loopback}
set portList {}

#####
# Chassis-loopback #####
chassis get "loopback"
set chassis [chassis cget -id]

##### Card Type : FCM GXM8 #####
set card 57
card setDefault
card config -clockSelect cardClockInternal 0
card config -txFrequencyDeviation
set retCode [card set $chassis $card]
switch $retCode \
    $::TCL_OK { \
        if { [card write $chassis $card] != $::TCL_OK } { \
            errorMsg "Error calling card write" \
            $chassis $card"; \
            set retCode $::TCL_ERROR; \
        } \
    } \
    $::ixTcl_notAvailable { \
        logMsg "One or more of the ports on this card is \
unavailable, please check ownership. Card settings not applied."; \
    } \
    default { \
}
```

```

        errorMsg "Error calling card set $chassis $card";
    }

##### Chassis-loopback Card-57 Port-1 #####
set port      1

port setFactoryDefaults $chassis $card $port
port config -speed                      8500
port config -duplex                     full
port config -flowControl                false
port config -directedAddress           "01 80 C2 00 00 01"
port config -multicastPauseAddress     "01 80 C2 00 00 01"
port config -loopback                  portNormal
port config -transmitMode              portTxPacketStreams
port config -receiveMode               [expr
$::portCapture|$::portRxDataIntegrity|$::portRxSequenceChecking|$::portRxModeWidePacketGroup]
port config -autonegotiate             false
port config -advertise100FullDuplex   false
port config -advertise100HalfDuplex   false
port config -advertise10FullDuplex    false
port config -advertise10HalfDuplex   false
port config -advertise1000FullDuplex  false
port config -portMode                 9
port config -enableDataCenterMode    false
port config -dataCenterMode          eightPriorityTrafficMapping
port config -flowControlType         ieee8023x
port config -pfceEnableValueListBitMatrix  ""
port config -pfcrResponseDelayEnabled 0
port config -pfcrResponseDelayQuanta 0
port config -rxTxMode                gigNormal
port config -ignoreLink              false
port config -advertiseAbilities     portAdvertiseNone
port config -timeoutEnable          true
port config -negotiateMasterSlave   0
port config -masterSlave            portSlave
port config -pmaClock
pmaClockAutoNegotiate
port config -enableSimulateCableDisconnect false
port config -enableAutoDetectInstrumentation true
port config -autoDetectInstrumentationMode portAutoInstrumentationModeFloating
port config -enableRepeatableLastRandomPattern 0
port config -transmitClockDeviation 0
port config -transmitClockMode       portClockInternal
port config -preEmphasis             preEmphasis0
port config -transmitExtendedTimestamp 0
port config -operationModeList      [list
$::portOperationModeStream]
port config -MacAddress              "00 de bb 00 00 01"
port config -DestMacAddress         "00 de bb 00 00 02"
port config -name                   ""
port config -numAddresses          1
port config -enableManualAutoNegotiate false
port config -enablePhyPolling       true
port config -enableTxRxSyncStatsMode false
port config -txRxSyncInterval      0
port config -enableTransparentDynamicRateChange false
port config -enableDynamicMPLSMode  false
port config -enablePortCpuFlowControl false

```

```

port config -portCpuFlowControlDestAddr          "01 80 C2 00 00 01"
port config -portCpuFlowControlSrcAddr          "00 00 01 00 02 00"
port config -portCpuFlowControlPriority         "0 0 0 0 0 0 0 0"
port config -portCpuFlowControlType              0
port config -enableWanIFSStretch                false
if {[port set $chassis $card $port]} {
    errorMsg "Error calling port set $chassis $card $port"
    set retCode $::TCL_ERROR
}

fcPort config -enableAutoNegotiate             false
fcPort config -bbCredit                         8
fcPort config -bbSCN                           0
fcPort config -forceErrorMode                 0
fcPort config -doNotSendRRDYAfterNFrames      0
fcPort config -rrdyResponseDelayMode          0
fcPort config -fixedDelayValue                0
fcPort config -eDTOVMode                      2
fcPort config -eDTOVOverride                  2000
fcPort config -rATOVMode                     2
fcPort config -rATOVOverride                  10000
fcPort config -rTTOVMode                      2
fcPort config -rTTOVOverride                  100
if {[fcPort set $chassis $card $port]} {
    errorMsg "Error calling fcPort set $chassis $card $port"
    set retCode $::TCL_ERROR
}

stat setDefault
stat config -mode                             statNormal
stat config -enableValidStats                false
stat config -enableProtocolServerStats       true
stat config -enableArpStats                  true
stat config -enablePosExtendedStats          true
stat config -enableDhcpStats                 false
stat config -enableDhcpV6Stats               false
stat config -enableFcoeStats                true
stat config -fcoeRxSharedStatType1          statFcoeValidFrames
stat config -fcoeRxSharedStatType2          statFcoeValidFrames
if {[stat set $chassis $card $port]} {
    errorMsg "Error calling stat set $chassis $card $port"
    set retCode $::TCL_ERROR
}

packetGroup setDefault
packetGroup config -signatureOffset           48
packetGroup config -signature                "08 71 18 05"
packetGroup config -insertSignature          false
packetGroup config -ignoreSignature          false
packetGroup config -groupId                  0
packetGroup config -groupIdOffset            52
packetGroup config -enableGroupIdMask        false
packetGroup config -enableInsertPgid         true
packetGroup config -groupIdMask              4293918720
packetGroup config -latencyControl           cutThrough
packetGroup config -measurementMode          packetGroupModeLatency
packetGroup config -delayVariationMode        delayVariationWithSequenceErrors
packetGroup config -preambleSize              8
packetGroup config -sequenceNumberOffset     44
packetGroup config -sequenceErrorThreshold   2

```

```

packetGroup config -insertSequenceSignature           false
packetGroup config -allocateUdf                     true
packetGroup config -enableSignatureMask            false
packetGroup config -signatureMask                 "00 00 00 00"
packetGroup config -enableRxFilter                false
packetGroup config -headerFilter                 "00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00"
packetGroup config -headerFilterMask              "00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00"
packetGroup config -enable128kBinMode             true
packetGroup config -enableTimeBins               false
packetGroup config -numPgidPerTimeBin           32
packetGroup config -numTimeBins                  1
packetGroup config -timeBinDuration            1000000
packetGroup config -enableLatencyBins           false
packetGroup config -latencyBinList               ""
packetGroup config -groupIdMode                 ""
packetGroupCustom                                seqThreshold
packetGroup config -sequenceCheckingMode        seqThreshold
packetGroup config -multiSwitchedPathMode       seqSwitchedPathPGID
packetGroup config -enableLastBitTimeStamp      false
if {[packetGroup setRx $chassis $card $port]} {
    errorMsg "Error calling packetGroup setRx $chassis $card
$port"
    set retCode $::TCL_ERROR
}

dataIntegrity setDefault                         40
dataIntegrity config -signatureOffset          "08 71 18
00"
dataIntegrity config -signature                false
dataIntegrity config -insertSignature          false
dataIntegrity config -enableTimeStamp         false
dataIntegrity config -floatingTimestampAndDataIntegrityMode
dataIntegrityNumberOfBytesFromEndOfFrame
dataIntegrity config -numBytesFromEndOfFrame   4
dataIntegrity config -payloadLength           0
if {[dataIntegrity setRx $chassis $card $port]} {
    errorMsg "Error calling dataIntegrity setRx $chassis
$card $port"
    set retCode $::TCL_ERROR
}

autoDetectInstrumentation setDefault
autoDetectInstrumentation config -startOfScan 0
autoDetectInstrumentation config -signature
{87 73 67 49 42 87 11 80 08 71 18 05}
autoDetectInstrumentation config -enableSignatureMask
false
autoDetectInstrumentation config -signatureMask
{00 00 00 00 00 00 00 00 00 00 00 00}
if {[autoDetectInstrumentation setRx $chassis $card $port]} {
    errorMsg "Error calling autoDetectInstrumentation setRx
$chassis $card $port"
    set retCode $::TCL_ERROR
}

linkFaultSignaling setDefault                   2
linkFaultSignaling config -contiguousErrorBlocks
linkFaultSignaling config -contiguousGoodBlocks 0

```

```

linkFaultSignaling config -sendSetsMode
linkFaultAlternateOrderedSets
linkFaultSignaling config -loopCount 1
linkFaultSignaling config -enableLoopContinuously true
linkFaultSignaling config -enableTxIgnoresRxLinkFault
false
linkFaultSignaling config -orderedSetTypeA
linkFaultLocal
linkFaultSignaling config -orderedSetTypeB
linkFaultRemote
if {[linkFaultSignaling set $chassis $card $port]} {
    errorMsg "Error calling linkFaultSignaling set $chassis
$card $port"
    set retCode $::TCL_ERROR
}

capture setDefault
capture config -fullAction lock
capture config -sliceSize 65536
capture config -sliceOffset 0
capture config -captureMode
captureTriggerMode
capture config -continuousFilter 0
capture config -beforeTriggerFilter
captureBeforeTriggerNone
capture config -afterTriggerFilter
captureAfterTriggerFilter
capture config -triggerPosition 1.0
capture config -enableSmallPacketCapture false
if {[capture set $chassis $card $port]} {
    errorMsg "Error calling capture set $chassis $card $port"
    set retCode $::TCL_ERROR
}

filter setDefault
filter config -captureTriggerDA anyAddr
filter config -captureTriggerSA anyAddr
filter config -captureTriggerPattern anyPattern
filter config -captureTriggerError errAnyFrame
filter config -captureTriggerFrameSizeEnable false
filter config -captureTriggerFrameSizeFrom 36
filter config -captureTriggerFrameSizeTo 36
filter config -captureTriggerCircuit filterAnyCircuit
filter config -captureFilterDA anyAddr
filter config -captureFiltersSA anyAddr
filter config -captureFilterPattern anyPattern
filter config -captureFilterError errAnyFrame
filter config -captureFilterFrameSizeEnable false
filter config -captureFilterFrameSizeFrom 36
filter config -captureFilterFrameSizeTo 36
filter config -captureFilterCircuit filterAnyCircuit
filter config -userDefinedStat1DA anyAddr
filter config -userDefinedStat1SA anyAddr
filter config -userDefinedStat1Pattern anyPattern
filter config -userDefinedStat1Error errAnyFrame
filter config -userDefinedStat1FrameSizeEnable false
filter config -userDefinedStat1FrameSizeFrom 36
filter config -userDefinedStat1FrameSizeTo 36
filter config -userDefinedStat1Circuit filterAnyCircuit
filter config -userDefinedStat2DA anyAddr
filter config -userDefinedStat2SA anyAddr
filter config -userDefinedStat2Pattern anyPattern

```

```

filter config -userDefinedStat2Error          errAnyFrame
filter config -userDefinedStat2FrameSizeEnable 0
filter config -userDefinedStat2FrameSizeFrom   36
filter config -userDefinedStat2FrameSizeTo     36
filter config -userDefinedStat2Circuit        filterAnyCircuit
filter config -asyncTrigger1DA                anyAddr
filter config -asyncTrigger1SA                anyAddr
filter config -asyncTrigger1Pattern          anyPattern
filter config -asyncTrigger1Error            errAnyFrame
filter config -asyncTrigger1FrameSizeEnable  false
filter config -asyncTrigger1FrameSizeFrom    36
filter config -asyncTrigger1FrameSizeTo      36
filter config -asyncTrigger1Circuit          filterAnyCircuit
filter config -asyncTrigger2DA                anyAddr
filter config -asyncTrigger2SA                anyAddr
filter config -asyncTrigger2Pattern          anyPattern
filter config -asyncTrigger2Error            errAnyFrame
filter config -asyncTrigger2FrameSizeEnable  false
filter config -asyncTrigger2FrameSizeFrom    36
filter config -asyncTrigger2FrameSizeTo      36
filter config -asyncTrigger2Circuit          filterAnyCircuit
filter config -captureTriggerEnable         true
filter config -captureFilterEnable          true
filter config -userDefinedStat1Enable       false
filter config -userDefinedStat2Enable       false
filter config -asyncTrigger1Enable          false
filter config -asyncTrigger2Enable          false
if {[filter set $chassis $card $port]} {
    errorMsg "Error calling filter set $chassis $card $port"
    set retCode $::TCL_ERROR
}

filterPallete setDefault                      "00 00 00
filterPallete config -DA1                     "00 00 00
00 00 00"
filterPallete config -DAMask1                 "00 00 00
00 00 00"
filterPallete config -DA2                     "00 00 00
00 00 00"
filterPallete config -DAMask2                 "00 00 00
00 00 00"
filterPallete config -SA1                     "00 00 00
00 00 00"
filterPallete config -SAMask1                 "00 00 00
00 00 00"
filterPallete config -SA2                     "00 00 00
00 00 00"
filterPallete config -SAMask2                 "00 00 00
00 00 00"
filterPallete config -pattern1                "DE ED EF
FE AC CA"
filterPallete config -patternMask1           "00 00 00
00 00 00"
filterPallete config -pattern2                00
filterPallete config -patternMask2           00
filterPallete config -patternOffset1         12
filterPallete config -patternOffset2         12
filterPallete config -matchType1             matchUser
filterPallete config -matchType2             matchUser
filterPallete config -patternOffsetType1    filterOffsetStartOfFrame
filterPallete config -patternOffsetType2    filterOffsetStartOfFrame
filterPallete config -patternOffsetType1    filterOffsetStartOfFrame

```

```

filterPalette config -gfpErrorCondition
gfpErrorsOr
filterPalette config -enableGfptHecError true
filterPalette config -enableGfpHecError true
filterPalette config -enableGfpPayloadCrcError true
filterPalette config -enableGfpBadFcsError true
filterPalette config -circuitList ""
if {[filterPalette set $chassis $card $port]} {
    errorMsg "Error calling filterPalette set $chassis $card
$port"
    set retCode $::TCL_ERROR
}

streamRegion setDefault
streamRegion config -gapControlMode
streamGapControlFixed
if {[streamRegion set $chassis $card $port]} {
    errorMsg "Error calling streamRegion set $chassis $card
$port"
    set retCode $::TCL_ERROR
}

ipAddressTable setDefault
ipAddressTable config -defaultGateway "0.0.0.0"
if {[ipAddressTable set $chassis $card $port]} {
    errorMsg "Error calling ipAddressTable set $chassis $card
$port"
    set retCode $::TCL_ERROR
}

if {[interfaceTable select $chassis $card $port]} {
    errorMsg "Error calling interfaceTable select $chassis
$card $port"
    set retCode $::TCL_ERROR
}

interfaceTable setDefault
interfaceTable config -dhcpV4RequestRate 0
interfaceTable config -dhcpV6RequestRate 0
interfaceTable config -dhcpV4MaximumOutstandingRequests 100
interfaceTable config -dhcpV6MaximumOutstandingRequests 100
interfaceTable config -fcoeRequestRate 500
interfaceTable config -fcoeNumRetries 5
interfaceTable config -fcoeRetryInterval 2000
interfaceTable config -fipVersion
fipVersion1
interfaceTable config -enableFcfcMac false
interfaceTable config -fcfcMacCollectionTime 1000
interfaceTable config -enablePMacInFpma true
interfaceTable config -enableNameIdInVLANDiscovery false
interfaceTable config -enableTargetLinkLayerAddrOption false
if {[interfaceTable set]} {
    errorMsg "Error calling interfaceTable set"
    set retCode $::TCL_ERROR
}

interfaceTable clearAllInterfaces

#### Interface entry type - interfaceTypeConnected
interfaceEntry clearAllItems addressTypeIpV6
interfaceEntry clearAllItems addressTypeIpV4

```

```

interfaceEntry setDefault

fcNameServer setDefault
fcNameServer config -enableRnnId          true
fcNameServer config -enableRcsId         false
fcNameServer config -enableRftId          true
fcNameServer config -enableRpnId          false
fcNameServer config -enableRptId          false
fcNameServer config -enableRspnId         false
fcNameServer config -enableRsnnNn        false
fcNameServer config -enableRhaid          false
fcNameServer config -symbolicPortName    ""
fcNameServer config -symbolicNodeName    ""
fcNameServer config -rhaid                ""

fcNameServerQuery setDefault
fcNameServerQuery config -fcNameServerQueryCommand
commandGANxt
fcNameServerQuery config -fcNameServerQueryObject
objectPortId
fcNameServerQuery config -fcNameServerQueryObjectValue      " " 

fcProperties removeAllPlogis
fcProperties setDefault
fcProperties config -sourcePortWWN        "10 00 00
00 96 C2 1A 16"
fcProperties config -sourceNodeWWN       "20 00 00
00 96 C2 1A 16"
fcProperties config -destinationId      "01.b6.69"
fcProperties config -sourceOui           "0e.fc.00"
fcProperties config -bufferToBufferRxSize 2112
fcProperties config -enableSCR          false
fcProperties config -enableNs            true
fcProperties config -enablePlogi         false
fcProperties config -enableAutoPlogi     true
fcProperties config -enableNSQuery       true
fcProperties config -enablePRLI          true
fcProperties config -scrOption          2

interfaceEntry config -enable             true
interfaceEntry config -description
{ProtocolInterface - 57:01 - 1}
interfaceEntry config -macAddress        {00 00 96
C2 1B EB}
interfaceEntry config -eui64Id           {02 00 96
FF FE C2 1B EB}
interfaceEntry config -mtu               1500
interfaceEntry config -enableDhcp        false
interfaceEntry config -enableVlan        false
interfaceEntry config -vlanId            0
interfaceEntry config -vlanPriority     0
interfaceEntry config -vlanTPID          0x8100
interfaceEntry config -enableDhcpV6      false
interfaceEntry config -ipV6Gateway      {0:0:0:0:0:0:0:0}
interfaceEntry config -enableFlogi        true
if {[interfaceTable addInterface interfaceTypeConnected]} {
    errorMsg "Error calling interfaceTable addInterface
interfaceTypeConnected"
    set retCode $::TCL_ERROR
}

```

```

if {[interfaceTable write]} {
    errorMsg "Error calling interfaceTable write"
    set retCode $::TCL_ERROR
}

protocolServer setDefault
protocolServer config -enableArpResponse false
protocolServer config -enablePingResponse false
if {[protocolServer set $chassis $card $port]} {
    errorMsg "Error calling protocolServer set $chassis $card
$port"
    set retCode $::TCL_ERROR
}

ixEnablePortIntrinsicLatencyAdjustment $chassis $card $port false
lappend portList [list $chassis $card $port]
ixWritePortsToHardware portList
ixCheckLinkState portList

#####
## Generating streams for all the ports from above
#####
#####

#####
##### Chassis-loopback Card-57 Port-1 #####
#####

chassis get "loopback"
set chassis [chassis cget -id]
set card 57
set port 1
streamRegion get $chassis $card $port
if {[streamRegion enableGenerateWarningList $chassis $card $port 0]} {
    errorMsg "Error calling streamRegion
enableGenerateWarningList $chassis $card $port 0"
    set retCode $::TCL_ERROR
}

set streamId 1

# Stream 1
stream setDefault
stream config -name ""
stream config -enable true
stream config -enableSuspend false
stream config -region 0
stream config -numBursts 1
stream config -numFrames 100
stream config -ifg 14.1176470588
stream config -ifgType gapFixed
stream config -ifgMIN 28.2352941176
stream config -ifgMAX 37.6470588235
stream config -ibg 28.2352768842
stream config -enableIbg false
stream config -isg 28.2352768842
stream config -enableIsq false

```

```

stream config -gapUnit gapNanoSeconds
stream config -percentPacketRate 100.0
stream config -fpsRate 3035714.28571
stream config -bpsRate 5828571428.57
stream config -rateMode usePercentRate
stream config -preambleSize 28
stream config -preambleData "55 55 55 55 55
55 D5"
stream config -framesize 240
stream config -frameSizeType sizeAuto
stream config -frameSizeMIN 240
stream config -frameSizeMAX 240
stream config -frameSizeStep 4
stream config -enableTimestamp false
stream config -fcs good
stream config -patternType incrByte
stream config -dataPattern x00010203
stream config -pattern "00 01 02 03"
stream config -frameType "FF FF"
stream config -dma contPacket
stream config -rxTriggerEnable false
stream config -asyncIntEnable true
stream config -loopCount 1
stream config -returnToId 1
stream config -enforceMinGap 12
stream config -enableStatistic true
stream config -enableIncrFrameBurstOverride false
stream config -enableDisparityError false
stream config -enableSourceInterface false
stream config -sourceInterfaceDescription ""
stream config -startTxDelayUnit $::startTxDelayBytes 4
stream config -startTxDelay 0.0
stream config -priorityGroup priorityGroup0

protocol setDefault
protocol config -name nativeFc
protocol config -appName noType
protocol config -ethernetType noType
protocol config -enable802dot1qTag vlanNone
protocol config -enableISLtag false
protocol config -enableMPLS false
protocol config -enableMacSec false
protocol config -enableOAM false
protocol config -enableProtocolPad

fcSOF setDefault
fcSOF config -startOfFrame fcSOFn3
if {[fcSOF set $chassis $card $port]} {
    errorMsg "Error calling fcSOF set $chassis $card $port"
    set retCode $::TCL_ERROR
}
fibreChannel setDefault
fibreChannel config -extHeaderDetails ""
fibreChannel config -destinationId "D1.D2.D3"
fibreChannel config -routingControlType fibreChannelDeviceDataFrames
fibreChannel config -routingControlInformation fibreChannelUncategorizedInformation
fibreChannel config -sourceId "8D.8E.8F"
fibreChannel config -csControlOrPriorityValue 0x00
fibreChannel config -frameControl "00 00 00"
fibreChannel config -type 0x20

```

```

fibreChannel config -sequenceCount 0
fibreChannel config -dataFieldControl 0x70
fibreChannel config -sequenceId 0x00
fibreChannel config -responderExchangeId "00 00"
fibreChannel config -originatorExchangeId "00 00"
fibreChannel config -parameter "00 00 00
00"
fibreChannel config -originatorExchangeCounter
fibreChannelIdle false
fibreChannel config -enableBadFibreChannelCrc false
fibreChannel config -enableUseFcControlBits
fibreChannel config -exchangeContext
fibreChannelOriginator
fibreChannel config -sequenceContext
fibreChannelInitiator
fibreChannel config -firstSequence
fibreChannelFirstSequenceOther
fibreChannel config -lastSequence
fibreChannelLastSequenceOther
fibreChannel config -endSequence
fibreChannelEndSequenceOther
fibreChannel config -endConnection
fibreChannelConnectionAlive
fibreChannel config -csControlOrPriority
fibreChannelCsCtl
fibreChannel config -sequenceInitiative
fibreChannelInitiativeHold
fibreChannel config -ackForm
fibreChannelOriginal
fibreChannel config -retransmittedSequence
fibreChannelOriginal
fibreChannel config -unidirectionalTransmit
fibreChannelBidirectional
fibreChannel config -continueSequenceCondition
fibreChannelNoInformation
fibreChannel config -abortSequenceCondition
fibreChannelContinue
fibreChannel config -relativeOffsetPresent
fibreChannelRelativeOffsetDefined
fibreChannel config -exchangeReassembly
fibreChannelExchangeReassemblyOff
fibreChannel config -fillBytes
fibreChannelZeroHexByteFill

if {[fibreChannel clearAllExtHeaders $chassis $card $port]} {
    errorMsg "Error calling fibreChannel clearAllExtHeaders
$chassis $card $port"
    set retCode $::TCL_ERROR
}

vftHeader setDefault
vftHeader config -virtualFabricId 0
vftHeader config -priority vftBestEffort
vftHeader config -type 0
vftHeader config -version 0
vftHeader config -routingControl 80
vftHeader config -hopCt 0
if {[fibreChannel addExtHeader extVFTHeader $chassis $card $port]} {
    errorMsg "Error calling fibreChannel addExtHeader
extVFTHeader $chassis $card $port"
    set retCode $::TCL_ERROR
}

```

```

ifrHeader setDefault
ifrHeader config -expirationTime          0
ifrHeader config -destinationFabricId    0
ifrHeader config -routingControl         81
ifrHeader config -hopCount               0
ifrHeader config -sourceFabricId        0
ifrHeader config -hopCountValid         0
ifrHeader config -expirationTimeValid   0
ifrHeader config -priority              0
ifrHeader config -version               0
if {[fibreChannel addExtHeader extIFRHeader $chassis $card $port]}
{
    errorMsg "Error calling fibreChannel addExtHeader
extIFRHeader $chassis $card $port"
    set retCode $::TCL_ERROR
}

encHeader setDefault
encHeader config -destinationId          0
encHeader config -routingControl         82
encHeader config -sourceId               0
encHeader config -csControlOrPriority   0
encHeader config -frameControl          0
encHeader config -type                  0
encHeader config -sequenceCount         0
encHeader config -dataFieldControl      0
encHeader config -sequenceId            0
encHeader config -responderExchangeId   0
encHeader config -originatorExchangeId  0
encHeader config -parameter             0
if {[fibreChannel addExtHeader extEncHeader $chassis $card $port]}
{
    errorMsg "Error calling fibreChannel addExtHeader
extEncHeader $chassis $card $port"
    set retCode $::TCL_ERROR
}

espHeader setDefault
espHeader config -espSequenceNumber      0
espHeader config -securityParameterIndex 0
if {[fibreChannel setOptHeader optESPHeader $chassis $card $port]}
{
    errorMsg "Error calling fibreChannel setOptHeader
optESPHeader $chassis $card $port"
    set retCode $::TCL_ERROR
}

associationHeader setDefault
associationHeader config -validity       192
associationHeader config -originatorProcessAssociator "00
00 00 00 00 00"
associationHeader config -responderProcessAssociator    "00
00 00 00 00 00"
if {[fibreChannel setOptHeader optAssociationHeader $chassis $card
$port]} {
    errorMsg "Error calling fibreChannel setOptHeader
optAssociationHeader $chassis $card $port"
    set retCode $::TCL_ERROR
}

IEEE48BitAddressDest setDefault

```


SEE ALSO

NAME - filter

filter - configure the filters of a port of a card on a chassis.

SYNOPSIS

filter *sub-command options*

DESCRIPTION

The *filter* command is used to configure the filters and capture triggers for receiving frames on a port of a card. The incoming frames can be filtered on a combination of varying constraints, such as destination or source address, pattern matching or specific error conditions.

Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in [atmHeader](#). The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command is with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

STANDARD OPTIONS

asyncTrigger1DA
true/false

Enables or disables User Defined Statistics counter 5 to filter on the destination MAC addresses. (*default = false*)

asyncTrigger1Enable
true/false

Enables or disables User Defined Statistics counter 5 that counts the number of frames filtered. To use this counter the **stat mode** has to be set to *statStreamTrigger*. (*default = false*)

asyncTrigger1Error
true/false

Enables or disables User Defined Statistics counter 5 filter on the errored frames. (*default = false*)

asyncTrigger1Frame
SizeEnable *true/false*

Enables or disables the frame size constraint which specifies a range of frame sizes to filter for User Defined Statistics counter 5. (*default = false*)

asyncTrigger1Frame
SizeFrom

The minimum range of the size of frame to be filtered for User Defined Statistics counter 5. Applicable only when **asyncTrigger1FramesizeEnable** is set to *true*. (*default = 64*)

asyncTrigger1Frame
SizeTo

The maximum range of the size of frame to be filtered for User Defined Statistics counter 5. Applicable only when **asyncTrigger1FramesizeEnable** is set to *true*. (*default = 1518*)

asyncTrigger1Pattern
true/false

Enables or disables User Defined Statistics counter 5 to filter on the pattern. (*default = false*)

asyncTrigger1SA
true/false

Enables or disables User Defined Statistics counter 5 to filter on the source MAC addresses. (*default = false*)

asyncTrigger2DA
true/false

Enables or disables User Defined Statistics counter 6 to filter on the destination MAC addresses. (*default = false*)

asyncTrigger2Enable <i>true/false</i>	Enables or disables User Defined Statistics counter 6 that counts the number of frames filtered. (<i>default = false</i>) To use this counter the stat mode has to be set to <i>statStreamTrigger</i> .																		
asyncTrigger2Error	Enables or disables User Defined Statistics counter 6 filter on the errored frames. (<i>default = false</i>)																		
asyncTrigger2FrameSizeEnable <i>true/false</i>	Enables or disables the frame size constraint which specifies a range of frame sizes to filter for User Defined Statistics counter 6. (<i>default = false</i>)																		
asyncTrigger2FrameSizeFrom	The minimum range of the size of frame to be filtered for User Defined Statistics counter 6. Applicable only when asyncTrigger1FramesizeEnable is set to <i>true</i> . (<i>default = 64</i>)																		
asyncTrigger2FrameSizeTo	The maximum range of the size of frame to be filtered for User Defined Statistics counter 6. Applicable only when asyncTrigger1FramesizeEnable is set to <i>true</i> . (<i>default = 1518</i>)																		
asyncTrigger2Pattern <i>true/false</i>	Enables or disables User Defined Statistics counter 6 to filter on the pattern. (<i>default = false</i>)																		
asyncTrigger2SA <i>true/false</i>	Enables or disables User Defined Statistics counter 6 to filter on the source MAC addresses. (<i>default = false</i>)																		
captureFilterDA	One of two available destination MAC addresses to filter on. Applicable only when <i>capturefilternable</i> is set to true. The possible values are:																		
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Option</th> <th style="text-align: center;">Value</th> <th style="text-align: center;">Usage</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;"><i>anyAddr</i></td> <td style="text-align: center;">0</td> <td>(<i>default</i>) disables the destination address filter constraint</td> </tr> <tr> <td style="text-align: center;"><i>addr1</i></td> <td style="text-align: center;">1</td> <td>sets the destination address filter constraint to trigger on frames with a destination MAC address that matches DA1 and DA1mask as specified in the filter palette</td> </tr> <tr> <td style="text-align: center;"><i>notAddr1</i></td> <td style="text-align: center;">2</td> <td>sets the destination address filter constraint to trigger on all frames except those with a destination MAC address that matches DA1 and DA1 mask as specified in the filter palette</td> </tr> <tr> <td style="text-align: center;"><i>addr2</i></td> <td style="text-align: center;">3</td> <td>sets the destination address filter constraint to trigger on frames with a destination MAC address that matches DA2 and DA2mask as specified in the filter palette</td> </tr> <tr> <td style="text-align: center;"><i>notAddr2</i></td> <td style="text-align: center;">4</td> <td>sets the destination address filter constraint to trigger on all frames except those with a destination MAC address that matches DA2 and DA2 mask as specified in the filter palette</td> </tr> </tbody> </table>		Option	Value	Usage	<i>anyAddr</i>	0	(<i>default</i>) disables the destination address filter constraint	<i>addr1</i>	1	sets the destination address filter constraint to trigger on frames with a destination MAC address that matches DA1 and DA1mask as specified in the filter palette	<i>notAddr1</i>	2	sets the destination address filter constraint to trigger on all frames except those with a destination MAC address that matches DA1 and DA1 mask as specified in the filter palette	<i>addr2</i>	3	sets the destination address filter constraint to trigger on frames with a destination MAC address that matches DA2 and DA2mask as specified in the filter palette	<i>notAddr2</i>	4	sets the destination address filter constraint to trigger on all frames except those with a destination MAC address that matches DA2 and DA2 mask as specified in the filter palette
Option	Value	Usage																	
<i>anyAddr</i>	0	(<i>default</i>) disables the destination address filter constraint																	
<i>addr1</i>	1	sets the destination address filter constraint to trigger on frames with a destination MAC address that matches DA1 and DA1mask as specified in the filter palette																	
<i>notAddr1</i>	2	sets the destination address filter constraint to trigger on all frames except those with a destination MAC address that matches DA1 and DA1 mask as specified in the filter palette																	
<i>addr2</i>	3	sets the destination address filter constraint to trigger on frames with a destination MAC address that matches DA2 and DA2mask as specified in the filter palette																	
<i>notAddr2</i>	4	sets the destination address filter constraint to trigger on all frames except those with a destination MAC address that matches DA2 and DA2 mask as specified in the filter palette																	
captureFilterEnable <i>true/false</i>	Enables or disables the capture filter. (<i>default = false</i>)																		
captureFilterError	Applicable only when <i>captureFilterEnable</i> is set to true. The possible values are:																		
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Option</th> <th style="text-align: center;">Value</th> <th style="text-align: center;">Usage</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;"><i>errAnyFrame</i></td> <td style="text-align: center;">0</td> <td>(<i>default</i>) disables the error filter constraint</td> </tr> </tbody> </table>		Option	Value	Usage	<i>errAnyFrame</i>	0	(<i>default</i>) disables the error filter constraint												
Option	Value	Usage																	
<i>errAnyFrame</i>	0	(<i>default</i>) disables the error filter constraint																	

Option	Value	Usage
<code>errGoodFrame</code>	1	sets the error filter constraint to trigger when frames with no errors are received
<code>errBadCRC</code>	2	sets the error filter constraint to trigger when frames with bad CRC errors are received
<code>errBadFrame</code>	3	sets the error filter constraint to trigger when corrupted frames are received
<code>errAlign</code>	4	sets the error filter constraint to trigger when frames with alignment errors are received (<i>10/100 only</i>)
<code>errDribble</code>	5	sets the error filter constraint to trigger when frames with dribble errors are received (<i>10/100 only</i>)
<code>errBadCRCAlignDribble</code>	5	sets the error filter constraint to trigger when frames with bad CRC, alignment error or dribble errors are received (<i>10/100 only</i>)
<code>errLineError</code>	4	sets the error filter constraint to trigger when frames with line errors are received (<i>gigabit only</i>)
<code>errLineAndBadCRC</code>	5	sets the error filter constraint to trigger line errors and bad CRC are received (<i>gigabit only</i>)
<code>errLineAndGoodCRC</code>	6	sets the error filter constraint to trigger when frames with line errors and bad CRC are received (<i>gigabit only</i>)
<code>errAnySequenceError</code>	4	sets the error filter constraint to trigger when any of the next three conditions are true
<code>errSmallSequenceError</code>	5	sets the error filter constraint to trigger when the current sequence number minus the previous sequence number is less than or equal to the error threshold and not negative, or when the current sequence number is equal to the previous sequence number
<code>errBigSequenceError</code>	6	sets the error filter constraint to trigger when the current sequence number minus the previous sequence number is greater than the error threshold
<code>errReverseSequenceError</code>	7	sets the error filter constraint to trigger when the current sequence number is less than the previous sequence number
<code>errDataIntegrityError</code>	8	sets the error filter constraint to trigger when any data integrity error is detected
<code>errGfpErrors</code>	9	sets the error filter constraint to trigger when any GFP error is detected. The particular errors that are used are controlled by options of the <code>filterPalette</code> command.
<code>errCdlErrors</code>	10	sets the error filter constraint to trigger when any CDL preamble error is detected
<code>errFcoeInvalidFrame</code>	12	sets the error filter constraint to trigger when any FCoE invalid frame error is detected
<code>errAnyIpTcpUdpChecksumError</code>	13	sets the error filter to trigger and filter on any Ip/Tcp/Udp checksum error

**captureFilterFrame
SizeEnable *true/false***

Enables or disables the frame size constraint which specifies a range of frame sizes to filter. (*default = false*)

captureFilterFrameSizeFrom

Applicable only when *captureFilterFrameSizeEnable* is enabled. The minimum range of the size of frame to be filtered. (*default* = 64)

captureFilterFrameSizeTo

Applicable only when *captureFilterFrameSizeEnable* is enabled. The maximum range of the size of frame to be filtered. (*default* = 1518)

captureFilterPattern

Applicable only when *captureFilterEnable* is set to true. The possible values are:

Option	Value	Usage
<i>anyPattern</i>	0	(<i>default</i>) disables the pattern filter constraint
<i>pattern1</i>	1	sets the pattern filter constraint to trigger on frames with a pattern that matches pattern1 and patternMask1 at offset patternOffset1 as specified in the filter palette
<i>notPattern1</i>	2	sets the pattern filter constraint to trigger on frames except those with a pattern that matches pattern1 and patternMask1 at offset patternOffset1 as specified in the filter palette
<i>pattern2</i>	3	sets the pattern filter constraint to trigger on frames with a pattern that matches pattern2 and patternMask2 at offset patternOffset2 as specified in the filter palette
<i>notPattern2</i>	4	sets the pattern filter constraint to trigger on frames except those with a pattern that matches pattern2 and patternMask2 at offset patternOffset2 as specified in the filter palette
<i>pattern1AndPattern2</i>	5	sets the pattern filter constraint to trigger on frames with a pattern that matches pattern1, pattern2 and patternMask1, patternMask2 at offset patternOffset1 and patternOffset2 as specified in the filter palette

captureFilterSA

One of two available destination MAC addresses to filter on. Applicable only when *capturefilternable* is set to true. The possible values are:

Option	Value	Usage
<i>anyAddr</i>	0	(<i>default</i>) disables the destination address filter constraint
<i>addr1</i>	1	sets the destination address filter constraint to trigger on frames with a destination MAC address that matches SA1 and DA1 mask as specified in the filter palette
<i>notAddr1</i>	2	sets the destination address filter constraint to trigger on all frames except those with a destination MAC address that matches SA1 and DA1 mask as specified in the filter palette
<i>addr2</i>	3	sets the destination address filter constraint to trigger on frames with a destination MAC address that matches SA2 and DA2 mask as specified in the filter palette
<i>notAddr2</i>	4	sets the destination address filter constraint to trigger on all frames except those with a destination MAC address that matches SA2 and DA2 mask as specified in the filter palette

captureTriggerDA	One of two available destination MAC addresses to filter on. Applicable only when <i>captureTriggerEnable</i> is set to true. The possible values are as in capturefilteredA . (<i>default = 0</i>)
captureTriggerEnable <i>true/false</i>	Enables or disables the capture trigger. (<i>default = false</i>)
captureTriggerError	Applicable only when <i>captureTriggerEnable</i> is set to true. The possible values are as in capturefilterrror . (<i>default = 0</i>)
captureTriggerFrame SizeEnable <i>true/false</i>	Enables or disables the frame size constraint which specifies a range of frame sizes to trigger. (<i>default = false</i>)
captureTriggerFrame SizeFrom	Applicable only when <i>captureTriggerFrameSizeEnable</i> is enabled. The minimum range of the size of frame to be triggered. (<i>default = 64</i>)
captureTriggerFrame SizeTo	Applicable only when <i>captureTriggerFrameSizeEnable</i> is enabled. The maximum range of the size of frame to be triggered. (<i>default = 1518</i>)
captureTriggerPattern	Applicable only when <i>captureTriggerEnable</i> is set to true. The possible values are as in captureFilterPattern . (<i>default = 0</i>)
captureTriggerSA	One of two available source MAC addresses to filter on. Applicable only when <i>captureTriggerEnable</i> is set to true. The possible values are as in captureFilterSA . (<i>default = 0</i>)
enableCircuitList <i>true/false</i>	Use the circuit list for filtering. (<i>default = false</i>)
userDefinedStat1DA	One of two available destination MAC addresses to filter on. Applicable only when <i>userDefinedStat1Enable</i> is set to true. The possible values are as in capturefilteredA . (<i>default = 0</i>)
userDefinedStat1 Enable <i>true/false</i>	Enables or disables the User Defined Statistics counter 1 that counts the number of frames filtered. (<i>default = false</i>)
userDefinedStat1Error	Applicable only when <i>userDefinedStat1Enable</i> is set to true. The possible values are as in capturefilterrror . (<i>default = 0</i>)
userDefinedStat1Frame SizeEnable <i>true/false</i>	Enables or disables the frame size constraint which specifies a range of frame sizes to count. (<i>default = false</i>)
userDefinedStat1Frame SizeFrom	Applicable only when <i>userDefinedStat1FrameSizeEnable</i> is enabled. The minimum range of the size of frame to be counted. (<i>default = 64</i>)
userDefinedStat1Frame SizeTo	Applicable only when <i>userDefinedStat1FrameSizeEnable</i> is enabled. The maximum range of the size of frame to be counted. (<i>default = 1518</i>)
userDefinedStat1 Pattern	Applicable only when <i>userDefinedStat1Enable</i> is set to true. The possible values are as in captureFilterPattern . (<i>default = 0</i>)
userDefinedStat1SA	One of two available source MAC addresses to filter on. Applicable only when <i>userDefinedStat1Enable</i> is set to true. The possible values are as in captureFilterSA . (<i>default = 0</i>)

userDefinedStat2DA	One of two available destination MAC addresses to filter on. Applicable only when <i>userDefinedStat2Enable</i> is set to true. The possible values are as in capturefilteredA . (<i>default = 0</i>)
userDefinedStat2Enable true/false	Enables or disables User Defined Statistics counter 2 that counts the number of frames filtered. (<i>default = false</i>)
userDefinedStat2Error	Applicable only when <i>userDefinedStat2Enable</i> is set to true. The possible values are as in capturefilterrror . (<i>default = 0</i>)
userDefinedStat2FrameSizeEnable true/false	Enables or disables the frame size constraint which specifies a range of frame sizes to count. (<i>default = false</i>)
userDefinedStat2FrameSizeFrom	Applicable only when <i>userDefinedStat2FrameSizeEnable</i> is enabled. The minimum range of the size of frame to be counted. (<i>default = 64</i>)
userDefinedStat2FrameSizeTo	Applicable only when <i>userDefinedStat2FrameSizeEnable</i> is enabled. The maximum range of the size of frame to be counted. (<i>default = 1518</i>)
userDefinedStat2Pattern	Applicable only when <i>userDefinedStat2Enable</i> is set to true. The possible values are as in captureFilterPattern . (<i>default = 0</i>)
userDefinedStat2SA	One of two available source MAC addresses to filter on. Applicable only when <i>userDefinedStat2Enable</i> is set to true. The possible values are as in captureFilterSA . (<i>default = 0</i>)

DEPRECATED OPTIONS

captureFilterError

The following *captureFilterError* options have been deprecated:

Option	Value	Usage
<i>errUndersize</i>	7	sets the error filter constraint to trigger when undersized frames (less than 64 bytes) are received
<i>errOversize</i>	8	sets the error filter constraint to trigger when oversized frames (greater than 1518 bytes) are received
<i>errFragment</i>	9	sets the error filter constraint to trigger when fragmented frames are received

COMMANDS

The **filter** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

filter cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **filter** command.

filter config option value

Modify the configuration options of the filter. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for filter.

filter get *chassisID cardID portID*

Gets the current configuration of the filter for port with id *portID* on card *cardID*, chassis *chassisID*. from its hardware. Call this command before calling **filter cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

filter set *chassisID cardID portID*

Sets the configuration of the filter in IxHAL for port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **filter config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

filter setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal
# In this example we will generate a range of packets with different frame sizes, DA/SA and
# data pattern in order to demonstrate how a directly attached port can collect specific
# statistics and trigger/filter on contents
set tclserver solarSystem
set host      galaxy
set username  user

# Check if we are running on UNIX - connect to the TCL Server
# Note: it is better to run the TCL Server on a pc other than your chassis, as it could
# potentially use up resources that the chassis needs.
if {[isUNIX]} {
    if {[ixConnectToTclServer $tclserver]} {
        errorMsg "Could not connect to TCL Server $tclserver"
        return 1
    }
}

# Now connect to the chassis
if {[ixConnectToChassis $host]} {
    errorMsg $::ixErrorInfo
    return $::TCL_ERROR
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assume card to be used is in slot 1
set card 1
set txPort 1
set rxPort 2
set portList [list [list $chas $card $txPort] \
                  [list $chas $card $rxPort] ]

# Login before taking ownership
ixLogin $username
```

```

# Take ownership of the ports to use
if {[ixTakeOwnership $portList]} {
    errorMsg "Error taking ownership"
    return $::TCL_ERROR
}

# Configure each port to factory defaults first, then configure the streams.
if {[setFactoryDefaults portList]} {
    errorMsg "Error - setFactoryDefaults failed"
    return $::TCL_ERROR
}

# Commit the port's phy configuration to hardware, then check the link state to make sure you
# come up in the proper speed setting. This may affect the stream rate for later
#configuration options.
ixWritePortsToHardware portList
ixCheckLinkState portList

# Configure the stream on the transmit port.
set streamId 1

stream setDefault
stream config -numFrames      100000
stream config -dma            stopStream
stream config -frameSizeType sizeRandom
stream config -sa              {00 00 00 01 01 01}
stream config -saRepeatCounter contIncrement
stream config -saMaskSelect   {FF FF FF FC FC FC}
stream config -da              {00 00 00 01 01 02}
stream config -daRepeatCounter contIncrement
stream config -daMaskSelect   {FF FF FF FC FC FC}
if {[stream set $chassis $card $txPort $streamId]} {
    errorMsg "Error - stream set $chassis $card $txPort $streamId failed"
    return $::TCL_ERROR
}

# Configure the filters on the receive port.
filter setDefault
filter config -captureTriggerDA      addr1
filter config -captureTriggerSA      addr1
filter config -captureTriggerEnable true
filter config -captureFilterPattern pattern1
filter config -captureFilterFrameSizeMode true
filter config -captureFilterFrameSizeFrom 128
filter config -captureFilterFrameSizeTo 1024
filter config -captureFilterEnable true
filter config -userDefinedStat1Enable true
filter config -userDefinedStat2Enable true
filter config -userDefinedStat1DA     addr1
filter config -userDefinedStat2SA     addr1
if {[filter set $chassis $card $rxPort]} {
    errorMsg "Error - filter set $chassis $card $rxPort failed"
    return $::TCL_ERROR
}

filterPallette setDefault
filterPallette config -DA1          {00 00 00 01 01 02}
filterPallette config -SA1          {00 00 00 01 01 01}
filterPallette config -pattern1    {02 02}
filterPallette config -patternMask1 {02 02}
if {[filterPallette set $chassis $card $rxPort]} {
    errorMsg "Error - filterPallette set $chassis $card $rxPort failed"
}

```

```
    return $::TCL_ERROR
}

# Here, we are committing just the stream and filter configuration. Because the PHY has
#already been configured, link state will not be affected.
ixWriteConfigToHardware portList

ixClearStats portList
ixStartPortCapture $chassis $card $rxPort
ixStartPortTransmit $chassis $card $txPort
# This delay is to allow the port to transmit for a little while before reading the stats.
after 1000

# This is a blocking call and will not return until transmit is complete.
ixCheckPortTransmitDone $chassis $card $txPort
ixStopPortCapture $chassis $card $rxPort

# Here you may retrieve the stats for both tx and rx ports at the same time, then compare.
ixRequestStats portList

# Retreive the total number of transmitted frames from the tx port.
if {[statList get $chassis $card $txPort]} {
    errorMsg "Error - statList get $chassis $card $txPort failed"
    return $::TCL_ERROR
}
set framesSent [statList cget -framesSent]

# Since we configured the capture filters to use the UDS stats, get them to compare to the
# transmit stats later on.
if {[statList get $chassis $card $rxPort]} {
    errorMsg "Error - statList get $chassis $card $rxPort failed"
    return $::TCL_ERROR
}

set userStat1 [statList cget -userDefinedStat1]
set userStat2 [statList cget -userDefinedStat2]
set triggered [statList cget -captureTrigger]

if {[capture get $chassis $card $rxPort]} {
    errorMsg "Error - capture get $chassis $card $rxPort failed"
    return $::TCL_ERROR
}

set captured [capture cget -nPackets]
ixPuts "frames sent: $framesSent"
ixPuts "$captured captured, $triggered triggered"
ixPuts "stat1 = $userStat1, stat2 = $userStat2"

# Let go of the ports that were reserved.
ixClearOwnership $portList
# Disconnect from the chassis in use.
ixDisconnectFromChassis $host

# If we are running on UNIX, disconnect from the TCL Server.
if [isUNIX] {
    ixDisconnectTclServer $tclserver
}

# This will cleanup any remaining memory, connections, etc. and should be called at the end
#of all scripts.
cleanUp
```

SEE ALSO

filterPallette

NAME - filterPalette

filterPalette - configure the filter palettes of a port on a card on a chassis.

SYNOPSIS	filterPalette <i>sub-command options</i>
DESCRIPTION	<p>The <i>filterPalette</i> command is used to configure the information that the receiving frames are going to be filtered on. This palette applies to all the filters (capture trigger, capture filter, user defined statistics 1 and 2) that are enabled by the <i>filter</i> command.</p> <p>When the setting for <i>filter captureFilterError</i> is set to <i>errGfpErrors</i>, the <i>enableGfpHecError</i>; <i>enableGfpeHecError</i>; <i>enableGfpPayloadCrcError</i> and <i>enableGfpBadFcsError</i> determine which GFP errors are used. The setting of <i>gfpErrorCondition</i> is used to determine if the OR or AND of these conditions are desired.</p> <p>For ports that support the <i>portFeaturePatternOffsetFlexible</i> feature, it is possible to specify the pattern offsets relative to the start of frame, start of IP frame, start of interior protocol or start of SONET frame.</p>
<hr/>	
STANDARD OPTIONS	
circuitList	Configure the list of circuits that would apply to filters. (<i>string</i>)
enableGfpBadFcsError <i>true / false</i>	If <i>true</i> , then GFP bad FCS errors are used in the filter. This condition is OR'd or AND'd with the other GFP errors based on the setting of the <i>gfpErrorCondition</i> option. (<i>default = true</i>)
enableGfpeHecError <i>true / false</i>	If <i>true</i> , then GFP extension header HEC errors are used in the filter. This condition is OR'd or AND'd with the other GFP errors based on the setting of the <i>gfpErrorCondition</i> option. (<i>default = true</i>)
enableGfpPayloadCrc Error <i>true / false</i>	If <i>true</i> , then GFP payload CRC errors are used in the filter. This condition is OR'd or AND'd with the other GFP errors based on the setting of the <i>gfpErrorCondition</i> option. (<i>default = true</i>)
enableGfptHecError <i>true / false</i>	If <i>true</i> , then GFP type header HEC errors are used in the filter. This condition is OR'd or AND'd with the other GFP errors based on the setting of the <i>gfpErrorCondition</i> option. (<i>default = true</i>)
DA1	Only frames that contain this destination MAC address are filtered, captured or counted. (<i>default = 00 00 00 00 00 00</i>)
DA2	Only frames that contain this destination MAC address are filtered, captured or counted. (<i>default = 00 00 00 00 00 00</i>)
DAMask1	A bit mask that allows to specify which bits of the <i>DA1</i> should be used when filtering. If the mask bit is set high, the pattern bit is used in the filter. (<i>default = 00 00 00 00 00 00</i>)

DAMask2

A bit mask that allows to specify which bits of the DA2 should be used when filtering. If the mask bit is set high, the pattern bit is used in the filter. (*default* = 00 00 00 00 00 00)

gfpErrorCondition

Indicates whether the enabled error conditions associated with *enableGptiHecError*, *enableGpeHecError*, *enableGfpPayloadCrcError* and *enableGfpBadFcsError* must all be present (AND'd) or only one must be present (OR).

Option	Value	Usage
<i>gfpErrorsOr</i>	0	(<i>default</i>) Only one of the enabled error conditions must be present.
<i>gfpErrorsAnd</i>	1	All of the enabled error conditions must be present.

matchType1

Match type for *pattern1* set in class member *pattern1*. The available match types are:

Option	Value	Usage
<i>matchIpEthernetII</i>	0	an Ethernet II packet.
<i>matchIp8023Snap</i>	1	an 802.3 SNAP packet.
<i>matchVlan</i>	2	a VLAN tagged packet.
<i>matchUser</i>	3	(<i>default</i>) a value as specified by pattern1, pattern Mask1 and patternOffset1.
<i>matchIpPPP</i>	4	a PPP format packet
<i>matchIpCiscoHDLC</i>	5	a Cisco HDLC format packet.
<i>matchIpSAEEthernetII</i>	6	match the IP Source Address for an Ethernet II packet located at offset 26.
<i>matchIpDAEEthernetII</i>	7	match the IP Destination Address for an Ethernet II packet located at offset 30.
<i>matchIpSADAEthernetII</i>	8	match the IP Source and Destination Address for an Ethernet II packet located at offset 26.
<i>matchIpSA8023Snap</i>	9	match the IP Source Address for an 802.3 Snap packet located at offset 34.
<i>matchIpDA8023Snap</i>	10	match the IP Destination Address for an 802.3 Snap packet located at offset 38.
<i>matchIpSADA8023Snap</i>	11	match the IP Source and Destination Address for an 802.3 Snap packet located at offset 34.
<i>matchIpSAPos</i>	12	match the IP Source Address for an POS packet located at offset 16.
<i>matchIpDAPos</i>	13	match the IP Destination Address for an POS packet located at offset 20.
<i>matchIpSADAPos</i>	14	match the IP Source and Destination Addresses for an POS packet located at offset 16.
<i>matchTcpSourcePortIPEthernetII</i>	15	match the TCP Source Port for an Ethernet II packet located at offset 34.

Option	Value	Usage
matchTcpDestPortIPEthernetII	16	match the TCP Destination Port for an Ethernet II packet located at offset 36.
matchUdpSourcePortIPEthernetII	17	match the UDP Source Port for an Ethernet II packet located at offset 34.
matchIpSAPos	12	match the IP Source Address for an POS packet located at offset 16.
matchIpDAPos	13	match the IP Destination Address for an POS packet located at offset 20.
matchIpSADAPos	14	match the IP Source and Destination Addresses for an POS packet located at offset 16.
matchTcpSourcePortIPEthernetII	15	match the TCP Source Port for an Ethernet II packet located at offset 34.
matchTcpDestPortIPEthernetII	16	match the TCP Destination Port for an Ethernet II packet located at offset 36.
matchUdpSourcePortIPEthernetII	17	match the UDP Source Port for an Ethernet II packet located at offset 34.
matchUdpDestPortIPEthernetII	18	match the UDP Destination Port for an Ethernet II packet located at offset 36.
matchTcpSourcePortIP8023Snap	19	match the TCP Source Port for an 802.3 Snap packet located at offset 42.
matchTcpDestPortIP8023Snap	20	match the TCP Destination Port for an 802.3 Snap packet located at offset 44.
matchUdpSourcePortIP8023Snap	21	match the UDP Source Port for an 802.3 Snap packet located at offset 42.
matchUdpDestPortIP8023Snap	(22)	match the UDP Destination Port for an 802.3 Snap packet located at offset 44
matchTcpSourcePortIPPos	23	match the TCP Source Port for a POS packet located at offset 24.
matchTcpDestPortIPPos	24	match the TCP Destination Port for a POS packet located at offset 26.
matchUdpSourcePortIPPos	25	match the UDP Source Port for a POS packet located at offset 24.
matchUdpDestPortIPPos	26	match the UDP Source Port for a POS packet located at offset 26
matchSrpModeReserved000	27	match an SRP packet whose mode is reserved 000.
matchSrpModeReserved001	28	match an SRP packet whose mode is reserved 001.
matchSrpModeReserved010	29	match an SRP packet whose mode is reserved 010.
matchSrpModeAtmCell011	30	match an SRP packet whose mode is ATM cell.
matchSrpControlMessagePassToHost100	31	match an SRP packet whose mode is control message 1.
matchSrpControlMessageBufferForHost101	32	match an SRP packet whose mode is control message 2.

Option	Value	Usage
matchSrpUsageMessage110	33	match an SRP packet which is an SRP usage message.
matchSrpPacketData111	34	match an SRP packet which is a data packet.
matchSrpAllControlMessages10x	35	match SRP control messages 1 and 2.
matchSrpUsageMessageOrPacketData11x	36	match SRP usage message or data packet.
matchSrpControlUsageOrPacketData1xx	37	match SRP usage message, control message 1 or 2, or data packet.
matchSrpInnerRing	38	match an SRP packet whose <i>ringIdentifier</i> is set to <i>inner</i> .
matchSrpOuterRing	39	match an SRP packet whose <i>ringIdentifier</i> is set to <i>outer</i> .
matchSrpPriority0-7	40-47	match an SRP packet whose <i>priority</i> is set to 0 - 7.
matchSrpParityOdd	48	match an SRP packet with odd parity.
matchSrpParityEven	49	match an SRP packet with even parity.
matchSrpDiscoveryFrame	50	match an SRP discovery packet.
matchSrplpsFrame	51	match an SRP IPS packet.
matchRprRingId0	52	Match any RPR packet which specifies Ringlet 0. (Originally transmitted on Ringlet 0 by the Source)
matchRprRingId1	53	Match any RPR packet which specifies Ringlet 1. (Originally transmitted on Ringlet 1 by the Source)
matchRprFairnessEligibility0	54	Match any RPR packet which specifies Fairness Eligibility 0. (0 = Not eligible for Fairness algorithm)
matchRprFairnessEligibility1	55	Match any RPR packet which specifies Fairness Eligibility 1. (0 = Not eligible for Fairness algorithm)
matchRprIdlePacket	56	Match any RPR Idle packet (Type = 00).
matchRprControlPacket	57	Match any RPR Control packet. (Type = 01)
matchRprFairnessPacket	58	Match any RPR Fairness packet. (Type = 10)
matchRprDataPacket	59	Match any RPR Data packet. (Type = 11)
matchRprServiceClassC	60	Match any RPR packet which specifies service Class C.
matchRprServiceClassB	61	Match any RPR packet which specifies service Class B.
matchRprServiceClassA1	62	Match any RPR packet which specifies service Class A1.
matchRprServiceClassA0	63	Match any RPR packet which specifies service Class A0.

Option	Value	Usage
matchRprWrapEligibility0\	64	Match any RPR packet which specifies Wrap Eligibility 0. (0 = Steerable only)
matchRprWrapEligibility1	65	Match any RPR packet which specifies Wrap Eligibility 1. (1 = Wrap Eligible)
matchRprParityBit0	66	Match any RPR packet which specifies Parity Bit 0.
matchRprParityBit1	67	Match any RPR packet which specifies Parity Bit 1.
matchIpv6SAEthernetII	68	Match the IPv6 Source Address for an Ethernet II packet.
matchIpv6DAEthernetII	69	Match the IPv6 Destination Address for an Ethernet II packet.
matchIpv6SA8023Snap	70	Match the IPv6 Source Address for an 802.3 packet.
matchIpv6DA8023Snap	71	Match the IPv6 Destination Address for an 802.3 packet.
matchIpv6SAPos	72	Match the IPv6 Source Address for a POS packet.
matchIpv6DAPos	73	Match the IPv6 Destination Address for a POS packet.
matchIpv6TcpSourcePort EthernetII	74	Match the IPv6 TCP source port number for an Ethernet II packet.
matchIpv6TcpDestPortEthernetII	75	Match the IPv6 TCP destination port number for an Ethernet II packet.
matchIpv6UdpSourcePort EthernetII	76	Match the IPv6 UDP source port number for an Ethernet II packet.
matchIpv6UdpDestPortEthernetII	77	Match the IPv6 UDP destination port number for an Ethernet II packet.
matchIpv6TcpSourcePort 8023Snap	78	Match the IPv6 TCP source port number for an 802.3 SNAP packet.
matchIpv6TcpDestPort8023Snap	79	Match the IPv6 TCP destination port number for an 802.3 Snap packet.
matchIpv6UdpSourcePort 8023Snap	80	Match the IPv6 UDP source port number for an 802.3 Snap packet.
matchIpv6UdpDestPort8023Snap	81	Match the IPv6 UDP destination port number for an 802.3 Snap packet.
matchIpv6TcpSourcePortPos	82	Match the IPv6 TCP source port number for an pos packet.
matchIpv6TcpDestPortPos	83	Match the IPv6 TCP destination port number for an pos packet.
matchIpv6UdpSourcePortPos	84	Match the IPv6 UDP source port number for an pos packet.
matchIpv6UdpDestPortPos	85	Match the IPv6 UDP destination port number for an pos packet.
matchIpv6IpTcpSourcePort EthernetII	86	Match the TCP source port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in an Ethernet II packet.

Option	Value	Usage
matchIpv6IpTcpDestPortEthernetII	87	Match the TCP destination port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in an Ethernet II packet.
matchIpv6IpUdpSourcePortEthernetII	88	Match the UDP source port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in an Ethernet II packet.
matchIpv6IpUdpDestPortEthernetII	89	Match the UDP destination port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in an Ethernet II packet.
matchIpv6IpTcpSourcePort8023Snap	90	Match the TCP source port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in an 802.3 Snap packet.
matchIpv6IpTcpDestPort8023Snap	91	Match the TCP destination port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in an 802.3 Snap packet.
matchIpv6IpUdpSourcePort8023Snap	92	Match the UDP source port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in an 802.3 Snap packet.
matchIpv6IpUdpDestPort8023Snap	93	Match the UDP destination port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in an 802.3 Snap packet.
matchIpv6IpTcpSourcePortPos	94	Match the TCP source port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in a POS packet.
matchIpv6IpTcpDestPortPos	95	Match the TCP destination port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in a POS packet.
matchIpv6IpUdpSourcePortPos	96	Match the UDP source port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in a POS packet.
matchIpv6IpUdpDestPortPos	97	Match the UDP destination port number for an IPv4 over IPv6 or IPv6 over IPv4 frame in a POS packet.
matchIpOverIpv6IpSAEthernetII	98	Match the IPv4 source address in an IPv4 frame encapsulated in an IPv6 frame in an Ethernet II packet.
matchIpOverIpv6IpDAEthernetII	99	Match the IPv4 destination address in an IPv4 frame encapsulated in an IPv6 frame in an Ethernet II packet.
matchIpOverIpv6IpSA8023Snap	100	Match the IPv4 source address in an IPv4 frame encapsulated in an IPv6 frame in an 802.3 Snap packet.
matchIpOverIpv6IpDA8023Snap	101	Match the IPv4 destination address in an IPv4 frame encapsulated in an IPv6 frame in an 802.3 Snap packet.
matchIpOverIpv6IpSAPos	102	Match the IPv4 source address in an IPv4 frame encapsulated in an IPv6 frame in POS packet.

Option	Value	Usage
matchIpOverIpv6IpDAPos	103	Match the IPv4 destination address in an IPv4 frame encapsulated in an IPv6 frame in POS packet.
matchIpv6OverIpv6SAEthernetII	104	Match the IPv6 source address in an IPv6 frame encapsulated in an IPv4 frame in an Ethernet II packet.
matchIpv6OverIpv6DAEthernetII	105	Match the IPv6 destination address in an IPv6 frame encapsulated in an IPv4 frame in an Ethernet II packet.
matchIpv6OverIpv6SA8023Snap	106	Match the IPv6 source address in an IPv6 frame encapsulated in an IPv4 frame in an 802.3 Snap packet.
matchIpv6OverIpv6DA8023Snap	107	Match the IPv6 destination address in an IPv6 frame encapsulated in an IPv4 frame in an 802.3 Snap packet.
matchIpv6OverIpv6SAPos	108	Match the IPv6 source address in an IPv6 frame encapsulated in an IPv4 frame in POS packet.
matchIpv6OverIpv6DAPos	109	Match the IPv6 destination address in an IPv6 frame encapsulated in an IPv4 frame in POS packet.
matchIpv6Ppp	110	Match an IPv6 PPP packet.
matchIpv6CiscoHdLC	111	Match an IPv6 packet encapsulated with Cisco HDLC.
matchGfpDataFcsNullExtEthernet	112	Match a user data GFP frame which includes an FCS and whose payload uses a null extension and indicates frame-mapped ethernet data.
matchGfpDataNoFcsNullExtEthernet	113	Match a user data GFP frame which does not includes an FCS and whose payload uses a null extension and indicates frame-mapped ethernet data.
matchGfpDataFcsLinearExtEthernet	114	Match a user data GFP frame which includes an FCS and whose payload uses a linear frame extension and indicates frame-mapped ethernet data.
matchGfpDataNoFcsLinearExtEthernet	115	Match a user data GFP frame which does not includes an FCS and whose payload uses a linear frame extension and indicates frame-mapped ethernet data.
matchGfpMgmtFcsNullExtEthernet	116	Match a management GFP frame which includes an FCS and whose payload uses a null extension and indicates frame-mapped ethernet data.
matchGfpMgmtNoFcsNullExtEthernet	117	Match a management GFP frame which does not includes an FCS and whose payload uses a null extension and indicates frame-mapped ethernet data.

Option	Value	Usage
matchGfpMgmtFcsLinearExtEthernet	118	Match a management GFP frame which includes an FCS and whose payload uses a linear frame extension and indicates frame-mapped ethernet data.
matchGfpMgmtNoFcsLinearExtEthernet	119	Match a management GFP frame which does not includes an FCS and whose payload uses a linear frame extension and indicates frame-mapped ethernet data.
matchGfpDataFcsNullExtPpp	120	Match a user data GFP frame which includes an FCS and whose payload uses a null extension and indicates frame-mapped PPP data.
matchGfpDataNoFcsNullExtPpp	121	Match a user data GFP frame which does not includes an FCS and whose payload uses a null extension and indicates frame-mapped PPP data.
matchGfpDataFcsLinearExtPpp	122	Match a user data GFP frame which includes an FCS and whose payload uses a linear frame extension and indicates frame-mapped PPP data.
matchGfpDataNoFcsLinearExtPpp	123	Match a user data GFP frame which does not includes an FCS and whose payload uses a linear frame extension and indicates frame-mapped PPP data.
matchGfpMgmtFcsNullExtPpp	124	Match a management GFP frame which includes an FCS and whose payload uses a null extension and indicates frame-mapped PPP data.
matchGfpMgmtNoFcsNullExtPpp	125	Match a management GFP frame which does not includes an FCS and whose payload uses a null extension and indicates frame-mapped PPP data.
matchGfpMgmtFcsLinearExtPpp	126	Match a management GFP frame which includes an FCS and whose payload uses a linear frame extension and indicates frame-mapped PPP data.
matchGfpMgmtNoFcsLinearExtPpp	127	Match a management GFP frame which does not includes an FCS and whose payload uses a linear frame extension and indicates frame-mapped PPP data.

matchType2

Match type for **pattern2**. The available match types are as in *matchType1*.
(default = 3)

pattern1

Only frames that contain this pattern at offset *patternOffset1* are filtered, captured or counted. *(default = “DE ED EF FE AC CA”)*

Note: Starting with IxOS 5.0, the hex string must be separated by a space between the hex bytes, for example: ‘00 80’.

pattern2	Only frames that contain this pattern at offset <i>patternOffset2</i> are filtered, captured or counted. (<i>default</i> = 00)										
patternMask1	A bit mask that allows to specify which bits of pattern1 should be used when filtering. If the mask bit is set low, the pattern bit is used in the filter. (<i>default</i> = 00)										
patternMask2	A bit mask that allows to specify which bits of pattern2 should be used when filtering. If the mask bit is set high, the pattern bit is used in the filter. (<i>default</i> = 00)										
patternOffset1	Offset of <i>pattern1</i> in the frame to be filtered, captured or counted. (<i>default</i> = 12)										
patternOffset2	Offset of <i>pattern2</i> in the frame to be filtered, captured or counted. (<i>default</i> = 12)										
patternOffsetType1	For ports that support the <i>portFeaturePatternOffsetFlexible</i> feature, this option specifies the place that <i>patternOffset1</i> is relative to. This value must be one of these options:										
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Option</th><th style="text-align: center;">Usage</th></tr> </thead> <tbody> <tr> <td style="text-align: center;"><i>filterPalletteOffsetStartOfFrame</i></td><td>(<i>default</i>) Offset from the start of the frame.</td></tr> <tr> <td style="text-align: center;"><i>filterPalletteOffsetStartOfIp</i></td><td>Offset from the start of the IP header</td></tr> <tr> <td style="text-align: center;"><i>filterPalletteOffsetStartOfProtocol</i></td><td>Offset from the start of the protocol within the IP header.</td></tr> <tr> <td style="text-align: center;"><i>filterPalletteOffsetStartOfSonet</i></td><td>Offset from the start of the SONET frame.</td></tr> </tbody> </table>		Option	Usage	<i>filterPalletteOffsetStartOfFrame</i>	(<i>default</i>) Offset from the start of the frame.	<i>filterPalletteOffsetStartOfIp</i>	Offset from the start of the IP header	<i>filterPalletteOffsetStartOfProtocol</i>	Offset from the start of the protocol within the IP header.	<i>filterPalletteOffsetStartOfSonet</i>	Offset from the start of the SONET frame.
Option	Usage										
<i>filterPalletteOffsetStartOfFrame</i>	(<i>default</i>) Offset from the start of the frame.										
<i>filterPalletteOffsetStartOfIp</i>	Offset from the start of the IP header										
<i>filterPalletteOffsetStartOfProtocol</i>	Offset from the start of the protocol within the IP header.										
<i>filterPalletteOffsetStartOfSonet</i>	Offset from the start of the SONET frame.										
patternOffsetType2	For ports that support the <i>portFeaturePatternOffsetFlexible</i> feature, this option specifies the place that <i>patternOffset1</i> is relative to. See <i>patternOffset</i>										
SA1	Only frames that contain this source MAC address are filtered, captured or counted. (<i>default</i> = 00 00 00 00 00 00)										
SA2	Only frames that contain this source MAC address are filtered, captured or counted. (<i>default</i> = 00 00 00 00 00 00)										
SAMask1	A bit mask that allows to specify which bits of the <i>SA1</i> should be used when filtering. If the mask bit is set high, the pattern bit is used in the filter. (<i>default</i> = 00 00 00 00 00 00)										
SAMask2	A bit mask that allows to specify which bits of the <i>SA2</i> should be used when filtering. If the mask bit is set high, the pattern bit is used in the filter. (<i>default</i> = 00 00 00 00 00 00)										

COMMANDS

The **filterPallette** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

filterPallette cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **filterPallette** command.

filterPallette config option value

Modify the configuration options of the filterPallette. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for filterPallette.

Note: Must be a valid Tcl list (values must be separated by spaces).

filterPallette get chassisID cardID portID

Gets the current config of the filterPallette on port *portID* on card *cardID*, chassis *chassisID*. from its hardware. Call this command before calling **filterPallette cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

filterPallette set chassisID cardID portID

Sets the configuration of the filterPallette in IxHAL on port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **filterPallette config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

filterPallette set chassisID cardID portID

Sets the configuration of the local port filterPallette object in IxHAL for port with id *portID* on card *cardID*, chassis *chassisID*. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

filterPallette setDefault

Sets to IxTclHal default values for all configuration options.

filterPallette write chassisID cardID portID

Writes or commits the changes in IxHAL to hardware for the filter palette on port with id *portID* on card *cardID*, chassis *chassisID*. Before using this command, use the **filterPallette set** command to configure the filterPallette related parameters in IxHAL. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- Network problem between the client and chassis

EXAMPLES

See examples under [filter](#).

SEE ALSO*filter*

NAME - fipTlv

fipTlv - describe/view a single FIP Tlv

SYNOPSIS	fipTlv <i>sub-command options</i>
-----------------	-----------------------------------

DESCRIPTION	The fipTlv command is used in two contexts:
	<ul style="list-style-type: none"> When a new TLV (type-length-value) is added to a <i>fcoeProperties</i> set. Values are taken from the options in this command. When an existing TLV is retrieved with <i>fcoeProperties get*Tlv</i>. The TLV values are visible in this command.

STANDARD OPTIONS

type	The type of the FIP Tlv option. (<i>default = 0</i>)
value	A string consisting of hexadecimal characters. Each pair of characters defines a byte value. The length of the TLV is set from the length of the value string, divided by 2. (<i>default = “”</i>)

COMMANDS	The <i>fipTlv</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	---

fipTlv **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *fipTlv* command.

fipTlv **config** *option value*

Modify the configuration options of the *fipTlv*. If no *option* is specified, returns a list describing all of the available options for *fipTlv* (see STANDARD OPTIONS).

fipTlv **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES	See example under <i>fcoeProperties</i> .
-----------------	---

SEE ALSO	<i>interfaceTable</i> , <i>interfaceEntry</i> , <i>fcoeProperties</i>
-----------------	---

NAME - flexibleTimestamp

flexibleTimestamp - configure the location of the time stamp in a packet

SYNOPSIS

flexibleTimestamp *sub-command options*

DESCRIPTION

The *flexibleTimestamp* command allows the placement of the packet time stamp value to be moved from its default place before the CRC to an offset within the packet. The availability of this feature for a particular port may be tested by use of the *port isValidFeature... portFeatureFlexibleTimestamp* command. Time stamps are inserted in transmitted packets by virtue of the *enableTimestamps* option in the *stream* command.

Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in *atmHeader*. The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command is with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

STANDARD OPTIONS

offset

If *type* is *timestampAtOffset*, then this is the offset within the packet to place the time stamp at. (*default = 23*)

type

The basic placement options.

Option	Value	Usage
<i>timestampBeforeCrc</i>	0	(<i>default</i>) Place the time stamp just before the CRC at the end of the packet.
<i>timestampAtOffset</i>	1	Place the time stamp at the offset indicated in <i>offset</i> .

COMMANDS

The **flexibleTimestamp** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

flexibleTimestamp **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **flexibleTimestamp** command.

flexibleTimestamp **config** *option value*

Modify the configuration options of the flexibleTimestamp. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for flexibleTimestamp.

flexibleTimestamp **get** *chasID cardID portID*

Gets the current configuration of the flexibleTimestamp header for port with id *portID* on card *cardID*, chassis *chasID* from its hardware. Call this command

before calling **flexibleTimestamp cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port does not support flexible time stamps

flexibleTimestamp set *chasID cardID portID*

Sets the flexibleTimestamp configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **flexibleTimestamp config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is owned by another user
- Configured parameters are not valid for this setting
- The port does not support flexible time stamps

flexibleTimestamp setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 18
set port 1
set portList [list [list $chas $card $port]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```
# See if the port supports flexible time stamps
if [port isValidFeature $chassis $card $port
portFeatureFlexibleTimestamp] {
    flexibleTimestamp config -type timestampAtOffset
    flexibleTimestamp config -offset 42
    if [flexibleTimestamp set $chassis $card $port] {
        ixPuts $::ixErrorInfo
        return 1
    }
    ixPuts "$chassis:$card:$port flexible time stamps set"
} else {
    ixPuts "$chassis:$card:$port does not support flexible time
stamps"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

port, stream

NAME - forcedCollisions

forcedCollisions - configure the forced collision parameters for 10/100 ports

SYNOPSIS	forcedCollisions <i>sub-command options</i>
DESCRIPTION	The <i>forcedCollisions</i> command is used to configure the forced collision parameters for 10/100Mbit ports. Forced collisions cause deliberate collisions for specified duty cycles.
STANDARD OPTIONS	
collisionDuration	The duration of each collision, measured in nibbles. (<i>default = 10</i>)
consecutiveCollisions	The number of consecutive collisions to generate at a time. Collisions take place on the first received packet after enabled. (<i>default = 4</i>)
consecutive Non-CollidingPackets	After each time that the number of programmed consecutive collisions have occurred this is the number of packets that is not modified. (<i>default = 4</i>)
continuous true / false	If <i>true</i> , the pattern of collisions and non-collisions is repeated indefinitely. (<i>default = true</i>)
enable true / false	Enables the generation of forced collisions. (<i>default = false</i>)
packetOffset	The offset from the beginning of packet active carrier sense (the beginning of the preamble) to the start of the collision, measured in nibbles. (<i>default = 64</i>)
repeatCount	If continuous operation is not selected, this value is the number of times that the pattern of collisions/non-collisions is repeated. (<i>default = 2</i>)
COMMANDS	
forcedCollisions cget option	The forcedCollisions command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
forcedCollisions config option value	Modifies the configuration options of the forcedCollisions. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for forcedCollisions.
forcedCollisions get chasID cardID portID	Gets the current configuration of the forcedCollisions header for port with id <i>portID</i> on card <i>cardID</i> , chassis <i>chasID</i> from its hardware. Call this command before calling forcedCollisions cget option value to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port does not support forced collisions

forcedCollisions set *chasID cardID portID*

Sets the forcedCollisions configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **forcedCollisions config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is owned by another user
- Configured parameters are not valid for this setting
- The port does not support forced collisions

forcedCollisions setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal
set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assume that card 1 is a 10/100 card
set card    1
set portA   1
set portB   2
# Set up mapping
map new    -type oneZone
map config -type oneZone
map add    $chas $card $portA  $chas $card $portB
map add    $chas $card $portB  $chas $card $portA
set portList [list [list $chas $card $portA] [list $chas $card
$portB]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```

# Set up both ports to 10Mbps and half duplex
port setDefault
port config -autonegotiate false
port config -duplex half
port config -speed 10
port set $chassis $card $portA
port set $chassis $card $portB

# Configure forced collisions
forcedCollisions setDefault
forcedCollisions config -enable 1
forcedCollisions config -consecutiveNonCollidingPackets 9
forcedCollisions set $chassis $card $portA
forcedCollisions set $chassis $card $portB

# Make the collision backoff algorithm try harder
collisionBackoff setDefault
collisionBackoff config -maxRetryCount 32
collisionBackoff set $chassis $card $portA
collisionBackoff set $chassis $card $portB

# Configure the streams to transmit at 50%
stream setDefault
stream config -percentPacketRate 50
stream config -rateMode usePercentRate
stream config -dma stopStream
stream config -numFrames 10000
stream set $chassis $card $portA 1
stream set $chassis $card $portB 1

# Write config to hardware, check the link state and clear
statistics
# Error checking omitted for brevity
ixWritePortsToHardware one2oneArray
after 1000
ixCheckLinkState one2oneArray
ixClearStats one2oneArray

# Start collisions
ixStartCollisions one2oneArray
# Make sure that ports don't attempt to transmit at the same
instant
ixStartStaggeredTransmit one2oneArray

ixCheckTransmitDone one2oneArray

ixCollectStats $portList collisions rxStats totals
ixPuts "$totals total collisions, port 1 = $rxStats(1,1,1), port 2
= $rxStats(1,1,2)"

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*portGroup*

NAME - frameRelay

frameRelay - configure the Frame Relay header for a Packet over Sonet frame

SYNOPSIS

frameRelay *sub-command options*

DESCRIPTION

The **frameRelay** command is used to configure the Frame Relay parameters.

Note: To configure the **frameRelay** parameters, **sonet config -header** needs to be configured for the right Frame Relay headers first. Note that *stream* get must be called before this command's get sub-command.

STANDARD OPTIONS

addressSize

Address length in the Frame Relay frame header. (*default* = 2)

beCN

Backward congestion notification bit in the Frame Relay address field. (*default* = 0)

commandResponse

Command or Response bit in the Frame Relay address field. (*default* = 0)

control

Control information. (*default* = 3)

counterMode

For multiple DLCIs where supported, this is the DLCI incrementing mode.
Options include:

Option	Value	Usage
frameRelayIncrement	0	Increment for the number of values set in <i>repeatCount</i> .
frameRelayContIncrement	1	Increment continuously.
frameRelayDecrement	2	Decrement for the number of values set in <i>repeatCount</i> .
frameRelayContDecrement	3	Decrement continuously.
frameRelayIdle	4	(<i>default</i>) Don't change the DLCI.
frameRelayRandom	5	Set the DLCI to random values.

discardEligibleBit

Discard eligible bit in the Frame Relay address field. (*default* = 0)

dLCI

DLCI core indicator bit in the Frame Relay address field. (*default* = 0)

dLCICoreValue

Frame Relay address field. (*default* = 0)

etherType

Ethertype of protocol in use. (*default* = 65535)

extentionAddress0

Extention address 0 bit in theFrame Relay address field. (*default* = 0)

extentionAddress1

Extention address 1 bit in theFrame Relay address field. (*default* = 1)

extentionAddress2

Extention address 2 bit in theFrame Relay address field. (*default* = 0)

extentionAddress3

Extention address 3 bit in theFrame Relay address field. (*default* = 0)

fecn	Forward congestion notification bit in the Frame Relay address field. (<i>default = 0</i>)
maskSelect	For multiple DLCIs where supported. The mask is applied to the DLCI value (as expressed in hexadecimal format). The mask length is defined by the number of bytes in the address - 2, 3, or 4 bytes of 2 nibbles each. X's, 1's, and 0's may be entered. An 'X' allows the defined DLCI hex character to be visible, and active. A '1' or a '0' masks the DLCI character with that value, so only the entered '1' or '0' is visible and active. (<i>default = {00 00 00 00 00 00}</i>)
maskValue	For multiple DLCIs where supported. The <i>dcli</i> option masked with the <i>maskSelect</i> value. Note: frameRelay on MSM10G and MSM2.5G port does not support DLCI maskValue, and the maskSelect is always forced to 0.
nlpid	Network layer protocol identifier to identify the type of upper-layer protocol transmitted in the frame. (<i>default = 255</i>)
repeatCount	For multiple DLCIs where supported. If <i>counterMode</i> is set to <i>frameRelayIncrement</i> or <i>frameRelayDecrement</i> , the number of times to change the DLCI value. (<i>default = 16</i>)

COMMANDS The **frameRelay** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

frameRelay cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **frameRelay** command.

frameRelay config *option value*

Modify the configuration options of the frameRelay. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for frameRelay.

frameRelay decode *capFrame chasID cardID portID [circuitID]*

Decodes a captured frame in the capture buffer and updates TcHal. Specific errors are:

- No connection to a chassis
- The captured frame is not a valid Frame Relay frame

frameRelay get *chasID cardID portID [circuitID]*

Gets the current configuration of the frameRelay header for port with id *portID* on card *cardID*, chassis *chasID* from its hardware. Note that *stream* get must be called before this command's get sub-command. Call this command before calling **frameRelay cget *option value*** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is not a Packet over Sonet port

frameRelay set *chasID cardID portID [circuitID]*

Sets the frameRelay configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **frameRelay config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- The port is not a Packet over Sonet port.

frameRelay setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that an OC48 POS card is in slot 18
set card 18
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the type of card and check if it's the correct type
set ifType [card getInterface $chas $card]
if {$ifType != $::interfaceOc48} {
    ixPuts "Card $card is not an OC48c POS card ($ifType)"
    return 1
}
```

```

port setFactoryDefaults $chas $card $port

# Need to set header type to Frame Relay
sonet setDefault
sonet config -interfaceType oc48
sonet config -header sonetFrameRelay2427
if [sonet set $chas $card 1] {
    ixPuts "Can't sonet set $chas:$card:1"
    return 1
}

stream setDefault
stream config -percentPacketRate 100.0
stream config -rateMode usePercentRate

# Set DLCI and BECN bit
frameRelay setDefault
frameRelay config -becn 1

# Set the DLCI address to 42 and enable incrementing DLCI's
# with a mask of F0 XX

frameRelay config -dlci 42
frameRelay config -repeatCount 16
frameRelay config -counterMode frameRelayIncrement
frameRelay config -maskSelect {FF 00}
frameRelay config -maskValue {F0 FF}

if [frameRelay set $chas $card $port] {
    ixPuts "Can't frameRelay set $chas:$card:$port"
    return 1
}

if [stream set $chas $card $port 1] {
    ixPuts "Stream set failed"
    return 1
}

ixWriteConfigToHardware portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

NAME - gfp

gfp - configure GFP framing parameters

SYNOPSIS

gfp *sub-command options*

DESCRIPTION

The *gfp* command is used to set all GFP framing parameters. The *enablePli* and *pli* options control the payload length indicator. The *payloadType* option controls the specification of the payload type. The inclusion and type of FCS is controlled by the *fcs* option. The channel ID is specified in the *channelId* option. HEC error insertion is controlled by the *coreHecErrors*, *typeHecErrors* and *extensionHecErrors* options.

STANDARD OPTIONS

channelId

The channel ID associated with management GFP frames. (*default = 0*)

coreHecErrors

Allows for insertion of core header errors.

Option	Value	Usage
gfpHecNone	0	(<i>default</i>) No errors.
gfpHec1Bit	1	One bit error.
gfpHecMultipleBits	2	Multiple bit errors.

enablePli true / false

If *true*, enables the inclusion of the payload length indicator in the core header. The value of the PLI is in the *pli* option. (*default = false*)

extensionHecErrors

Allows for the configuration of extension header error correction.

Option	Value	Usage
gfpHecErrorsNone	0	(<i>default</i>) No errors.
gfpHecErrors1Bit	1	1 bit error.
gfpHecErrors2Bits	2	2 bit errors.
gfpHecErrors3Bits	3	3 bit errors.
gfpHecErrors4Bits	4	4 bit errors.
gfpHecErrors5Bits	5	5 bit errors.
gfpHecErrors6Bits	6	6 bit errors.
gfpHecErrors7Bits	7	7 bit errors.
gfpHecErrors8Bits	8	8 bit errors.
gfpHecErrors9Bits	9	9 bit errors.
gfpHecErrors10Bits	10	10 bit errors.
gfpHecErrors11Bits	11	11 bit errors.
gfpHecErrors12Bits	12	12 bit errors.
gfpHecErrors13Bits	13	13 bit errors.
gfpHecErrors14Bits	14	14 bit errors.
gfpHecErrors15Bits	15	15 bit errors.
gfpHecErrors16Bits	16	16 bit errors.

fcs

The frame check sequence (FCS) configuration.

Option	Value	Usage
gfpNoFcs	0	Do not include an FCS.
gfpGoodFcs	1	(default) Include a good FCS.
gfpBadFcs	2	Include a bad FCS.

payloadType

The type of data that is included in the payload.

Option	Value	Usage
gfpDataFcsNullExtensionEthernet	0x1001	(default) Ethernet data packet with FCS and no extension header.
gfpDataNoFcsNullExtensionEthernet	0x0001	Ethernet data packet with no FCS and no extension header.
gfpDataFcsLinearExtensionEthernet	0x1101	Ethernet data packet with FCS and linear extension header.
gfpDataNoFcsLinearExtensionEthernet	0x0101	Ethernet data packet with no FCS and linear extension header.
gfpMgmtFcsNullExtensionEthernet	0x3001	Ethernet management packet with FCS and no extension header.
gfpMgmtNoFcsNullExtensionEthernet	0x2001	Ethernet management packet with no FCS and no extension header.
gfpMgmtFcsLinearExtensionEthernet	0x3101	Ethernet management packet with FCS and linear extension header.
gfpMgmtNoFcsLinearExtensionEthernet	0x2101	Ethernet management packet with no FCS and linear extension header.
gfpDataFcsNullExtensionPpp	0x1002	PPP Data packet with FCS and no extension header.
gfpDataNoFcsNullExtensionPpp	0x0002	PPP Data packet with no FCS and no extension header.
gfpDataFcsLinearExtensionPpp	0x1102	PPP Data packet with FCS and linear extension header.
gfpDataNoFcsLinearExtensionPpp	0x0102	PPP Data packet with no FCS and linear extension header.
gfpMgmtFcsNullExtensionPpp	0x3002	PPP Management packet with FCS and no extension header.
gfpMgmtNoFcsNullExtensionPpp	0x2002	PPP Management packet with no FCS and no extension header.
gfpMgmtFcsLinearExtensionPpp	0x3102	PPP Management packet with FCS and linear extension header.
gfpMgmtNoFcsLinearExtensionPpp	0x2102	PPP Management packet with no FCS and linear extension header.

pli

If the value of *enablePli* is *true*, this is the value of the PLI. (default = 0)

typeHecErrors

Allows for the configuration of type header error correction.

Option	Value	Usage
gfpHecErrorsNone	0	(default) No errors.
gfpHecErrors1Bit	1	1 bit error.
gfpHecErrors2Bits	2	2 bit errors.
gfpHecErrors3Bits	3	3 bit errors.
gfpHecErrors4Bits	4	4 bit errors.
gfpHecErrors5Bits	5	5 bit errors.
gfpHecErrors6Bits	6	6 bit errors.
gfpHecErrors7Bits	7	7 bit errors.
gfpHecErrors8Bits	8	8 bit errors.
gfpHecErrors9Bits	9	9 bit errors.
gfpHecErrors10Bits	10	10 bit errors.
gfpHecErrors11Bits	11	11 bit errors.
gfpHecErrors12Bits	12	12 bit errors.
gfpHecErrors13Bits	13	13 bit errors.
gfpHecErrors14Bits	14	14 bit errors.
gfpHecErrors15Bits	15	15 bit errors.
gfpHecErrors16Bits	16	16 bit errors.

typeIdentifier

If the value of *enablePli* is *true*, this is the value of the PLI. (default = 0)

COMMANDS

The *gfp* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

gfp cget option

Returns the current value of the configuration option *gfp* by *option*. *Option* may have any of the values accepted by the *gfp* command, subject to the setting of the *enableValidStats* option.

gfp config option value

Modify the configuration options of the time server. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *gfp*.

gfp decode capFrame chasID cardID portID [circuitID]

Decodes a captured frame in the capture buffer and makes the data available in the STANDARD OPTIONS through *gfp cget*. The *capFrame* parameter must be obtained through a call to **stream packetview**.

If *circuitID* = 0, gets information for the port; if *circuitID* not 0, gets information for the circuit. Specific errors are:

- No connection to a chassis
- Invalid port number
- The captured frame is not a valid gfp frame

gfp get chasID cardID portID [circuitID]

Gets the current preamble configuration of the port with circuit *circuitID*, id *portID* on card *cardID*, chassis *chasID*. Call this command before calling *gfp*

cget *option* to get the value of the configuration option. If *circuitID* = 0, gets information for the port; if *circuitID* not 0, gets information for the circuit.

gfp set *chasID cardID portID [circuitID]*

Sets the preamble configuration of the port with circuit *circuitID*, id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the *gfp config option value* command. If *circuitID* = 0, gets information for the port; if *circuitID* not 0, gets information for the circuit.

gfp setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chassId [ixGetChassisID $host]

set cardId 37
set portId 1
set portList [list]

if { [port isValidFeature $chassId $cardId $portId
      $::portFeatureGfp] } {
    lappend portList [list $chassId $cardId $portId]
} else {
    errorMsg "Port doesn't support portFeatureGfp"
    return "FAIL"
}

sonet setDefault
sonet config -header           $::sonetGfp
sonet config -interfaceType   $::oc48

if {[sonet set $chassId $cardId $portId]} {
    ixPuts $::ixErrorInfo
    return "FAIL"
}

filterPallette config -gfpErrorCondition $::gfpErrorsOr
if {[filterPallette set $chassId $cardId $portId]} {
    ixPuts $::ixErrorInfo
    return "FAIL"
```

```
}
```

```
gfpOverhead setDefault
gfpOverhead config -deltaSyncState
$::gfpSyncStateK8
gfpOverhead config -enableSingleBitErrorCorrection $::true
gfpOverhead config -enablePayloadScrambling $::true

if {[gfpOverhead set $chassId $cardId $portId]} {
    ixPuts $::ixErrorInfo
    return "FAIL"
}

set streamId          1
stream   setDefault
stream   config -name   "gfp_stream"

gfp setDefault
gfp config -enablePli      $::true
gfp config -pli           12
gfp config -payloadType   $::gfpMgmtFcsNullExtensionEthernet
gfp config -fcs           $::gfpGoodFcs
gfp config -channelId     11
gfp config -coreHecErrors $::gfpCHecMultipleBits
gfp config -typeHecErrors $::gfpHecErrors2Bits
gfp config -extensionHecErrors $::gfpHecErrors10Bits

if {[gfp set $chassId $cardId $portId]} {
    ixPuts $::ixErrorInfo
    return "FAIL"
}

if {[stream set $chassId $cardId $portId $streamId]} {
    ixPuts $::ixErrorInfo
    return "FAIL"
}

ixWriteConfigToHardware      portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

sonet, gfpOverhead

NAME - gfpOverhead

gfpOverhead - configure additional GFP parameters

SYNOPSIS

gfpOverhead *sub-command options*

DESCRIPTION

The *gfpOverhead* command is used to set several operation parameters.

STANDARD OPTIONS

enablePayload Scrambling true / false Enables the use of payload scrambling. The payload is scrambled using the $x^{43} + 1$ algorithm. (*default = true*)

enableSingleBitError Correction true / false Enables the use of single bit error correction. (*default = true*)

deltaSyncState The number of cHEC matches (+1) needed to move the state machine from the *hunt* state to the *sync* state.

Option	Value	Usage
gfpSyncStateK1	0	(<i>default</i>) 1
gfpSyncStateK2	1	2
gfpSyncStateK3	2	3
gfpSyncStateK4	3	4
gfpSyncStateK5	4	5
gfpSyncStateK6	5	6
gfpSyncStateK7	6	7
gfpSyncStateK8	7	8

gfpCrc

Read-only. The calculated GFP CRC value.

COMMANDS

The *gfpOverhead* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

gfpOverhead cget option

Returns the current value of the configuration option *gfpOverhead* by *option*. *Option* may have any of the values accepted by the *gfpOverhead* command, subject to the setting of the *enableValidStats* option.

gfpOverhead config option value

Modify the configuration options of the time server. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *gfpOverhead*.

gfpOverhead get chasID cardID portID [circuitID]

Gets the current preamble configuration of the circuit *circuitID* on port *portID*, on card *cardID*, on chassis *chasID*. Call this command before calling *gfpOverhead cget option* to get the value of the configuration option. If *circuitID*

= 0, gets information for the port; if *circuitID* not 0, gets information for the circuit.

gfpOverhead **set *chassisID cardID portID circuitID***

Sets the preamble configuration of the circuit *circuitID* on port *portID*, on card *cardID*, on chassis *chassisID* by reading the configuration option values set by the *gfpOverhead config option value* command. If *circuitID* = 0, gets information for the port; if *circuitID* not 0, gets information for the circuit.

gfpOverhead **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *gfp*

SEE ALSO

sonet, gfp

NAME - `gre`

`gre` - configure GRE parameters

SYNOPSIS	<code>gre sub-command options</code>
DESCRIPTION	The <code>gre</code> command is used to set GRE operation parameters.
<hr/>	
STANDARD OPTIONS	
enableChecksum <code>true / false</code>	Enables the GRE checksum when set to <code>true</code> . (<i>default = true</i>)
enableKey<code>true / false</code>	Enables the GRE authentication key when set to <code>true</code> , (<i>default = true</i>)
enableSequence Number <code>true / false</code>	Enables the GRE sequence number option when set to true. (<i>default = true</i>)
EnableValidChecksum <code>true / false</code>	Setting this value to <i>True</i> ensures the GRE checksum value is a valid value, and returns a “Good” packet evaluation. (<i>default = true</i>)
key	The GRE key is an authentication key used by the receiving router to validate the GRE packets. This check box allows to edit the GRE key.
protocolType <code>IPv4 / IPv6</code>	Sets the protocol type.
reserved0	Sets the Reserved 0 bits in the GRE header.
reserved1	Sets the Reserve 1 bits in the GRE header
sequenceNumber	The Sequence Number is used by the receiving router to establish the order in which packets have been transmitted. This option allows to set the sequence number bits.
version	Sets the version of GRE used. GRE headers are organized differently and contain varying information, depending on the version number. (<i>default = </i>)
<hr/>	
COMMANDS	The <code>gre</code> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
gre cget <i>option</i>	<i>gre cget option</i> Returns the current value of the configuration option <code>gre</code> by <i>option</i> . <i>Option</i> may have any of the values accepted by the <code>gre</code> command.
gre config <i>option value</i>	<i>gre config option value</i> Modify the configuration options of GRE. If no <i>option</i> is specified, returns a list describing all of the available options for GRE.

gre get *chasID cardID portID*

Gets the current GRE configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Call this command before calling *gre cget option* to get the value of the configuration option.

gre set *chasID cardID portID*

Sets the GRE configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the *gre config option value* command.

gre setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package req IxTclHal

set hostnameloopback

if {[ixConnectToChassis $hostname]} {
    errorMsg "error connecting $hostname chassis"
    return "FAIL"
}

set chassId [chassis cget -id]
set cardId2
set portId1
set portList [list [list $chassId $cardId $portId] ]

set streamId 1

stream setDefault
stream config -name "ipv6 gre"
stream config -framesize 200

protocol setDefault
protocol config -name ipV6
protocol config -ethernetType ethernetII

ip setDefault
ip config -ipProtocol
ipV4ProtocolTcp
ip config -sourceIpAddr "112.1.1.1"
ip config -sourceIpMask "255.0.0.0"
if {[ip set $chassId $cardId $portId]} {
    errorMsg "Error setting ip on $chassId $cardId $portId."
}

tcp setDefault
tcp config -offset 5
tcp config -sourcePort 10
if {[tcp set $chassId $cardId $portId]} {
    errorMsg "Error setting tcp on $chassId $cardId $portId."
}

gre setDefault
gre config -enableKey true
gre config -enableSequenceNumber true
gre config -enableChecksum true
gre config -enableValidChecksum false
```

```

gre config -key          "aa 22 33 45"
gre config -sequenceNumber "ab c1 ab c1"
gre config -version      1
gre config -reserved0    "01 ee"
gre config -reserved1    "ab c3"
# this will configure gre encapsulation ip protocol
gre config -protocolType "08 00"
if {[gre set $chassisId $cardId $portId]} {
    errorMsg "Error setting gre on $chassisId $cardId $portId."
}

ipV6 setDefault
ipV6 config -sourceAddr "4444:4444:4444:4444:4444:444:0:0"
ipV6 config -nextHeader ipV6Routing

ipV6 clearAllExtensionHeaders

ipV6Routing setDefault
ipV6Routing config -reserved"00 00 00 00"
ipV6Routing config -nodeList"0:0:0:0:0:0:0:0"
ipV6 addExtensionHeader ipV6Routing
ipV6 addExtensionHeader ipV4ProtocolGre

if {[ipV6 set $chassisId $cardId $portId]} {
    errorMsg "Error setting ipV6 on $chassisId $cardId $portId."
}

if [stream set $chassisId $cardId $portId $streamId] {
    errorMsg "Error setting stream on port $chassisId $cardId $portId $streamId"
}

ixWriteConfigToHardware portList

if [stream get $chassisId $cardId $portId $streamId] {
    errorMsg "Error getting stream on port $chassisId $cardId $portId $streamId"
}

# This will get the outer IP configurations
if {[ipV6 get $chassisId $cardId $portId]} {
    errorMsg "Error getting ipV6 on $chassisId $cardId $portId."
}
ixPuts "sourceAddr: [ipV6 cget -sourceAddr]

# This will get the outer IP configurations
if {[gre get $chassisId $cardId $portId]} {
    errorMsg "Error getting gre on $chassisId $cardId $portId."
}
ixPuts "key: [gre cget -key]

if {[ip get $chassisId $cardId $portId]} {
    errorMsg "Error getting ip on $chassisId $cardId $portId."
}
ixPuts "ip: [ip cget -sourceIpAddr]

```

SEE ALSO

N/A

NAME - hdlc

hdlc - configure the HDLC header for a Packet over Sonet frame

SYNOPSIS *hdlc sub-command options*

DESCRIPTION The *hdlc* command is used to configure the HDLC parameters.

STANDARD OPTIONS

address The one-byte address field of the HDLC header used in conjunction with Packet over Sonet. Defined values include:

Option	Value	Usage
<i>pppAddress</i>	0xff	(default)
<i>ciscoAddress</i>	0x0f	

control The one-byte control field of the HDLC header used in conjunction with Packet over Sonet. Defined values include:

Option	Value	Usage
<i>pppControl</i>	0x03	(default)
<i>ciscoControl</i>	0x00	

protocol The two-byte protocol field of the HDLC header used in conjunction with Packet over Sonet. Defined values include:

Option	Value	Usage
<i>ppplp</i>	0x0021	(default)
<i>ciscolp</i>	0x0800	
<i>ciscolpV6</i>	0x86dd	
<i>pppPaddingProtocol</i>	0x0001	
<i>pppOSI</i>	0x0023	
<i>pppXeroxIDP</i>	0x0025	
<i>pppDECnet</i>	0x0027	
<i>pppAppletalk</i>	0x0029	
<i>pppIPX</i>	0x002b	
<i>pppCompressedTCPIP</i>	0x002d	
<i>pppUncompressedTCPIP</i>	0x002f	
<i>pppBPDU</i>	0x0031	
<i>pppSTII</i>	0x0033	
<i>pppBanyanVines</i>	0x0035	
<i>pppAppleTalkEDDP</i>	0x0039	
<i>pppAppleTalkSmartBuffered</i>	0x003b	
<i>pppMultiLink</i>	0x003d	
<i>pppFirstChoiceCompression</i>	0x00fd	

Option	Value	Usage
pppHelloPackets	0x0201	
pppIBMSourceRoutingBPDU	0x0203	
pppLuxcom	0x0231	
pppSigmaNetworkSystems	0x0233	
pppIPControlProtocol	0x8021	
pppOSIControlProtocol	0x8023	
pppXeroxIDPControlProtocol	0x8025	
pppDECnetControlProtocol	0x8027	
pppAppletalkControlProtocol	0x8029	
pppIPXControlProtocol	0x802b	
pppBridgingNCP	0x8031	
pppMultiLinkControlProtocol	0x803d	
pppComprControlProtocol	0x80fd	
pppLinkControlProtocol	0xc021	
pppPasswordAuthProtocol	0xc023	
pppLinkQualityReport	0xc025	

COMMANDS

The **hdlc** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

hdlc cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **hdlc** command.

hdlc config option value

Modify the configuration options of the hdlc. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for hdlc.

hdlc decode capFrame chasID cardID portID [circuitID]

Decodes a captured frame in the capture buffer and makes the data available in the STANDARD OPTIONS through **hdlc** cget. The *capFrame* parameter must be obtained through a call to **stream** packetview. Specific errors are:

- No connection to a chassis
- Invalid port number
- The captured frame is not a valid Hdlc frame
- The port is not a Packet over Sonet port.

hdlc get chasID cardID portID [circuitID]

Gets the current configuration of the hdlc header for port with id *portID* on card *cardID*, chassis *chasID* from its hardware. Call this command before calling **hdlc** cget *option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is not a Packet over Sonet port.

hdlc set *chasID cardID portID [circuitID]*

Sets the hdlc configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **hdlc config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The configured parameters are not valid for this port
- The port is not a Packet over Sonet port.

hdlc setCisco *protocolType chasID cardID portID*

Sets the configuration of the hdlc header to *ciscoAddress* and *ciscoControl* in IxHAL for port with id *portID* on card *cardID*, chassis *chasID*. Specific errors are:

- No connection to a chassis
- Invalid port number
- The configured parameters are not valid for this port
- The port is not a Packet over Sonet port.
- The *protocolType* is not one of *ciscoIp* or *ciscoIpV6*.

hdlc setDefault

Sets to IxTclHal default values for all configuration options.

hdlc setPpp *protocolType chasID cardID portID*

Sets the configuration of the hdlc header to *pppAddress* and *pppControl* in IxHAL for port with id *portID* on card *cardID*, chassis *chasID*. Specific errors are:

- No connection to a chassis
- Invalid port number
- The configured parameters are not valid for this port
- The port is not a Packet over Sonet port.
- The *protocolType* is not *pppIp*.

EXAMPLES

```
package require IxTclHal

set addressByte($::pppAddress)      "pppAddress"
set addressByte($::ciscoAddress)    "ciscoAddress"

set controlByte($::pppControl)      "pppControl"
set controlByte($::ciscoControl)    "ciscoControl"

set protocolByte($::pppIp)          "pppIp"
set protocolByte($::ciscoIp)        "cisco"
set protocolByte($::pppPaddingProtocol)
"pppPaddingProtocol"
set protocolByte($::pppOSI)          "pppOSI"
set protocolByte($::pppXeroxIDP)    "pppXeroxIDP"
set protocolByte($::pppDECnet)      "pppDECnet"
set protocolByte($::pppAppletalk)    "pppAppletalk"
set protocolByte($::pppIPX)          "pppIPX"
set protocolByte($::pppCompressedTCPIP)
"pppCompressedTCPIP"
```

```

set protocolByte($::pppUncompressedTCPIP)
"pppUncompressedTCPIP"
set protocolByte($::pppBPDU)                                "pppBPDU"
set protocolByte($::pppSTII)                                "pppSTII"
set protocolByte($::pppBanyanVines)                          "pppBanyanVines"
set protocolByte($::pppAppleTalkEDDP)                        "pppAppletalkEDDP"
set protocolByte($::pppMultiLink)                            "pppMultiLink"
set protocolByte($::pppFirstChoiceCompression)
"pppFirstChoiceCompression"
set protocolByte($::pppHelloPackets)                         "pppHelloPackets"
set protocolByte($::pppIBMSourceRoutingBPDU)
"pppIBMSourceRoutingBPDU"
set protocolByte($::pppLuxcom)                               "pppLuxcom"
set protocolByte($::pppSigmaNetworkSystems)
"pppSigmaNetworkSystems"
set protocolByte($::pppIPControlProtocol)
"pppIPControlProtocol"
set protocolByte($::pppOSIControlProtocol)
"pppOSIControlProtocol"
set protocolByte($::pppXeroxIDPControlProtocol)
"pppXeroxIDPControlProtocol"
set protocolByte($::pppDECnetControlProtocol)
"pppDECnetControlProtocol"
set protocolByte($::pppAppletalkControlProtocol)
"pppAppletalkControlProtocol"
set protocolByte($::pppIPXControlProtocol)
"pppIPXControlProtocol"
set protocolByte($::pppBridgingNCP)                           "pppBridgingNCP"
set protocolByte($::pppMultiLinkControlProtocol)
"pppMultiLinkControlProtocol"
set protocolByte($::pppComprControlProtocol)
"pppComprControlProtocol"
set protocolByte($::pppLinkControlProtocol)
"pppLinkControlProtocol"
set protocolByte($::pppPasswordAuthProtocol)
"pppPasswordAuthProtocol"
set protocolByte($::pppLinkQualityReport)
"pppPasswordAuthProtocol"

proc printOptions {} \
{
    set addr      [hdlc cget -address]
    set cntrl     [hdlc cget -control]
    set protocol  [hdlc cget -protocol]
#    ixPuts "address $addressByte($addr), control
$controlByte($cntrl), \
#                      protocol $protocolByte($protocol)"
    ixPuts "address $addr, control $cntrl, protocol $protocol"
}

# Connect to chassis and get chassis ID
set host galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

```

```

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that a POS card is in slot 2
set card 2

set portList [list [list $chas $card 1] [list $chas $card 2]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Check for missing card
if {[card get $chas $card] != 0} \
{
    ixPuts "Card $card does not exist"
    break
}

# Get the type of card and check if it's the correct type
set cardType [card cget -type]
if {$cardType != $::cardPOS2Port} \
{
    ixPuts "Card $card is not an 2 port POS card"
    exit
}

# Set the options to default values
hdhc setDefault
ixWriteConfigToHardware portList

# Get the current hdlc state from the cards
hdhc get $chas $card 1
printOptions

# Set to Cisco values
hdhc setCisco ciscoIp $chas $card 1
ixWritePortsToHardware portList
set x [hdhc cget -address]
if {"0x$x" == $::ciscoAddress} {
    ixPuts "OK"
} else {
    ixPuts "NG"
}
ixStartPortCapture $chas $card 1
ixStartPortTransmit $chas $card 2
after 2000
capture get $chas $card 1
captureBuffer get $chas $card 1 1 1
captureBuffer getframe 1
set frameData [captureBuffer cget -frame]

```

```
# Now have hdlc decode the header
hdlc decode $frameData $chassis $card 1
printOptions

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ppp

NAME - icmp

icmp - configure the ICMP parameters for a port on a card on a chassis

SYNOPSIS	icmp <i>sub-command options</i>
-----------------	---------------------------------

DESCRIPTION	The <i>icmp</i> command is used to configure the ICMP-specific information used when building ICMP-type packets. Note that <i>stream get</i> must be called before this command's <i>get</i> sub-command.
--------------------	---

STANDARD OPTIONS

checksum	<i>Read-only</i> Value of the checksum in the valid icmp stream. Valid only if the stream set is performed.
-----------------	--

code	Code for each type of message. (<i>default = 0</i>)
-------------	---

id	ID for each ping command; that is, for the <i>echoRequest</i> . (<i>default = 0</i>)
-----------	--

sequence	Sequence number for each ping command (sequence number for the <i>echoRequest</i>) (<i>default = 0</i>)
-----------------	--

type	<i>Read-only</i> The type of ICMP message to be sent. Options are:
-------------	--

Option	Value	Usage
<i>echoReply</i>	0	(<i>default</i>) when echo message is received (when IP address is valid and receiving side supports the requested functions)
<i>destUnreachable</i>	3	when a datagram cannot reach its destination
<i>sourceQuench</i>	4	when gateway does not have the buffer space needed to queue the datagrams
<i>redirect</i>	5	when the gateway and the host identified by the internet source address of the datagram are on the same network
<i>echoRequest</i>	8	when network connection is to be tested (by ping command test the validity of IP address)
<i>timeExceeded</i>	11	when time to live field is 0
<i>parameterProblem</i>	12	when there is a problem with the header parameters
<i>timeStampRequest</i>	13	to request the timestamp of the receipt at the other end
<i>timeStampReply</i>	14	to get the timestamp when the datagram began its return
<i>infoRequest</i>	15	when host needs to find out the number of the network it is on.
<i>infoReply</i>	16	when <i>infoRequest</i> is received
<i>maskRequest</i>	17	to get the subnet address mask from the router
<i>maskReply</i>	18	when <i>maskRequest</i> is received

COMMANDS

The **icmp** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmp cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **icmp** command.

icmp config option value

Modify the ICMP configuration options of the port. If no *option* is specified, returns a list describing all of the available ICMP options (see STANDARD OPTIONS) for port.

icmp decode capFrame [chassisID cardID portID]

Decodes a captured frame in the capture buffer and updates TclHal. **icmp cget option** command can be used after decoding to get the option data. Specific errors are:

- No connection to a chassis
- Invalid port number
- The captured frame is not a valid Icmp frame

icmp get chassisID cardID portID

Gets the current ICMP configuration of the port with id *portID* on card *cardID*, chassis *chassisID*. Note that *stream* get must be called before this command's get sub-command. Call this command before calling **icmp cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

icmp set chassisID cardID portID

Sets the ICMP configuration of the port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **icmp config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

icmp setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

# In this example we'll send an echo response message from a port
# back to itself and decode the received packet

set host 400-031561
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
```

```

if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assume card to be used is in slot 1
set card 1
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Some defines for IP setup
set portMAC {00 00 00 01 01 01}
set portIP {192.168.18.1}
set portMask {255.255.255.0}

set destMAC {00 00 00 01 01 02}
set destIP {192.168.18.2}
set destMask {255.255.255.0}

# Put the port in loopback mode
port setFactoryDefaults $chassis $card $port
port setDefault
port config -loopback true

# Stream: 1 packet at 1%
stream setDefault
stream config -numFrames 1
stream config -dma stopStream
stream config -rateMode usePercentRate
stream config -percentPacketRate 1

# set protocol to IP
protocol setDefault
protocol config -name ip
protocol config -etherType ethernetII

# Set up IP: icmp with 46 byte packet
ip setDefault
ip config -ipProtocol icmp
ip config -totalLength 46
ip config -sourceIpAddr $portIP

```

```

ip config -sourceIpMask          $portMask
ip config -sourceClass           classC
ip config -destIpAddr            $destIP
ip config -destIpMask            $destMask
ip config -destClass             classC
ip set $chas $card $port

# Send an echo reply with some data in id and sequence
icmp setDefault
icmp config -type                echoReply
icmp config -code                0
icmp config -id                  3
icmp config -sequence             42
icmp set $chas $card $port

stream set $chas $card $port 1
port set $chas $card $port

# Set up the port
ixWritePortsToHardware portList

# Start capture and send the packet
after 1000
ixStartPortCapture $chas $card $port
ixStartPortTransmit $chas $card $port

# Stop port capture
after 1000
ixStopPortCapture $chas $card $port

# Get the capture buffer
captureBuffer get $chas $card $port
if {[captureBuffer cget -numFrames] == 0} {
    ixPuts "No packets received"
} else {
    # Get the frame
    captureBuffer getframe 1
    set data [captureBuffer cget -frame]

    # And decode the data
    icmp decode $data $chas $card $port
    ixPuts -nonewline "Received packet: code = "
    ixPuts -nonewline [icmp cget -code]
    ixPuts -nonewline ", id = "
    ixPuts -nonewline [icmp cget -id]
    ixPuts -nonewline ", sequence = "
    ixPuts           [icmp cget -sequence]
}
# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*stream, ip, udp*

NAME - icmpV6

icmpV6 - configure the ICMPv6 parameters for a port on a card on a chassis

SYNOPSIS

icmpV6 *sub-command options*

DESCRIPTION

The *icmpV6* command is used to define the ICMPv6 header type. ICMPv6 is used by IPv6 nodes to report errors encountered in processing packets, and to perform other internet-layer functions, such as diagnostics (ICMPv6 "ping"). ICMPv6 is an integral part of IPv6 and MUST be fully implemented by every IPv6 node.

IcmpV6 messages are grouped into classes:

- error messages: *icmpV6Error*
- informational messages: *icmpV6Informational*
- multicast listener discovery messages: *icmpV6MulticastListener*
- neighbor discovery messages: *icmpV6NeighborDiscovery*

STANDARD OPTIONS

type

Read-only. The type of ICMPv6 message to be sent. Options are:

Option	Value	Usage
<i>icmpV6DestUnreachableMessage</i>	1	(default) when a destination is unreachable
<i>icmpV6PacketTooBig Message</i>	2	when a packet is too big
<i>icmpV6TimeExceeded Message</i>	3	when hop limit is exceeded in transit
<i>icmpV6ParameterProblemMessage</i>	4	when erroneous header field is encountered
<i>icmpV6EchoRequest Message</i>	128	when network connection is to be tested (by ping command test the validity of IP address)
<i>icmpV6EchoReply Message</i>	129	when <i>echoRequest</i> is received
<i>icmpV6MulticastListenerQueryMessage</i>	130	multicast listener query
<i>icmpV6MulticastListenerReportMessage</i>	131	multicast listener report
<i>icmpV6MulticastListenerDoneMessage</i>	132	multicast listener done
<i>icmpV6RouterSolicitationMessage</i>	133	router solicitation
<i>icmpV6RouterAdvertisementMessage</i>	134	router advertisement
<i>icmpV6NeighborSolicitationMessage</i>	135	neighbor solicitation
<i>icmpV6NeighborAdvertisementMessage</i>	136	neighbor advertisement

Option	Value	Usage
<i>icmpV6RedirectMessage</i>	137	when the gateway and the host identified by the internet source address of the datagram are on the same network

COMMANDS

The **icmpV6** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6 cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **icmpV6** command.

icmpV6 config option value

Modify the ICMPv6 configuration options of the port. If no *option* is specified, returns a list describing all of the available ICMPv6 options for port (see STANDARD OPTIONS).

icmpV6 decode capFrame [chassisID cardID portID circuitID]

Decodes a captured frame in the capture buffer and updates TclHal. **icmpV6 cget option** command can be used after decoding to get the option data. Specific errors are:

- No connection to a chassis
- Invalid port number
- The captured frame is not a valid Icmpv6 frame

icmpV6 get chassisID cardID portID

Gets the current ICMPv6 configuration of the port with id *portID* on card *cardID*, chassis *chassisID*. **Note that stream get must be called before this command's get sub-command.** Call this command before calling **icmpV6 cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- Protocol data for this port is not yet set

icmpV6 set chassisID cardID portID

Sets the ICMPv6 configuration of the port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **icmpV6 config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

icmpV6 setDefault

Sets to IxTclHal default values for all configuration options.

icmpV6 setType messageType

Sets the message type. See the standard option **type** on page A-271 for a complete list.

EXAMPLES

```
package req IxTclHal

chassis add loopback

set chasId      [chassis cget -id]
set cardId      2
set portId      3
set streamId1

stream setDefault
stream config -framesize 200

# Configure protocol
protocol setDefault
protocol config -nameipV6
protocol config -etherTypeethernetII

icmpV6 setDefault
icmpV6 setType icmpV6RouterAdvertisementMessage

icmpV6NeighborDiscovery setDefault
icmpV6NeighborDiscovery config -currentHopLimit3
icmpV6NeighborDiscovery config -enableManagedAddressConfig $::true
icmpV6NeighborDiscovery config -enableOtherStatefulConfig $::true
icmpV6NeighborDiscovery config -routerLifetime10
icmpV6NeighborDiscovery config -reachableTime100
icmpV6NeighborDiscovery config -retransTimer1000

icmpV6OptionPrefixInformation setDefault
icmpV6OptionPrefixInformation config -length100
icmpV6OptionPrefixInformation config -prefixLength200
icmpV6OptionPrefixInformation config -enableLinkFlag $::true
icmpV6OptionPrefixInformation config -enableAutonomousAddressConfig $::true
icmpV6OptionPrefixInformation config -enableRouterAddress $::true
icmpV6OptionPrefixInformation config -enableSitePrefix $::true
icmpV6OptionPrefixInformation config -preferredLifetime2
icmpV6OptionPrefixInformation config -prefix"1111:1111:11:0:0:0:0:2"

if {[icmpV6NeighborDiscovery addOption icmpV6OptionPrefixInformation]} {
    ixPuts "Error addOption icmpV6OptionPrefixInformation (icmpV6OptionPrefixInformation) on
port $chasId.$cardId.$portId"
}

if {[icmpV6 set $chasId $cardId $portId]} {
    ixPuts "Error setting icmpV6 on port $chasId.$cardId.$portId"
}

# Configure IPv6
ipV6 setDefault
ipV6 config -trafficClass3
ipV6 config -sourceAddr           {1:2:3:0:0:0:0:0}
ipV6 config -sourceMask          64
ipV6 config -sourceAddrMode     ipV6Idle
ipV6 config -sourceStepSize     1
```

```
ipV6 config -sourceAddrRepeatCount10
ipV6 config -destAddr {4:5:6:0:0:0:0:0}

# Clear all the extension headers
ipV6 clearAllExtensionHeaders

# Add ipV4ProtocolIpv6Icmp
if {[ipV6 addExtensionHeader ipV4ProtocolIpv6Icmp]} {
    ixPuts "Error adding ipV4ProtocolIpv6Icmp"
}

if {[ipV6 set $chassisId $cardId $portId]} {
    ixPuts "Error setting ipV6 on port $chassisId.$cardId.$portId"
}

# Set and write the stream
if {[stream set $chassisId $cardId $portId $streamId]} {
    ixPuts "Error setting stream $streamId on port $chassisId.$cardId.$portId"
}

if {[stream write $chassisId $cardId $portId $streamId]} {
    ixPuts "Error writing stream $streamId on port $chassisId.$cardId.$portId"
}
```

SEE ALSO

*icmpV6Error, icmpV6Informational, icmpV6MulticastListener,
icmpV6NeighborDiscovery, icmpV6OptionLinkLayerDestination,
icmpV6OptionLinkLayerSource, icmpV6OptionMaxTransmissionUnit,
icmpV6OptionPrefixInformation, icmpV6OptionRedirectedHeader,
icmpV6OptionUserDefine, icmpV6UserDefine*

NAME - icmpV6Error

icmpV6error - configures the type and code (sub-type) of error message

SYNOPSIS

icmpV6error *sub-command options*

DESCRIPTION

The *icmpV6error* command is used to configure the type and code (sub-type) of error message to send.

STANDARD OPTIONS

type

Read-only The type of ICMPv6 message to be sent. The messageType must first be set in the *icmpV6* command by calling **setType**.

code

This parameter configures the code.

For *icmpV6DestinationUnreachable* code, options are:

Option	Value	Usage
<i>icmpV6NoRouteToDestination</i>	0	(<i>default</i>) there is no route to a destination
<i>icmpV6CommunicationProhibited</i>	1	communication prohibited
<i>icmpV6NotAssigned</i>	2	(not assigned)
<i>icmpV6AddressUnreachable</i>	3	address is unreachable
<i>icmpV6PortUnreachable</i>	4	port is unreachable
<i>icmpV6SourceAddressFailed</i>	5	address has failed
<i>icmpV6RejectRouteDestination</i>	6	route destination is rejected

For *icmpV6PacketTooBigCodeType*, this is always set to 0.

For *icmpV6TimeExceeded* code, options are:

Option	Value	Usage
<i>icmpV6HopLimitExceeded</i>	0	(<i>default</i>) hop limit was exceeded in transit
<i>icmpV6FragmentReassemblyTimeExceeded</i>	1	fragment reassembly time was exceeded

For *icmpV6ParameterProblemCodeType*; options are:

Option	Value	Usage
<i>icmpV6ErroneousHeaderFieldDetected</i>	0	(<i>default</i>) erroneous header field encountered
<i>icmpV6UnrecognizedNextHeaderType</i>	1	unrecognized Next Header type encountered
<i>icmpV6UnrecognizedIpv6Option</i>	2	unrecognized IPv6 option encountered

checkSum

Read-only. The 16-bit ICMPv6 checksum. This is the ones complement of the ones complement sum of the whole ICMPv6 message, which starts at the message type field. The 'whole' message includes the IPv6 header and extension header fields.

mtu Maximum Transmission Unit. Applies to *icmpV6PacketTooBigMessage* type only. The maximum size of the message that can be sent on this link to the next hop. (*default = 0*)

pointer Applies to *icmpV6ParameterProblemMessage* type only. It identifies the offset (octet) where the error was detected in the packet.

COMMANDS The **icmpV6error** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6error setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES See example under *icmpV6*

SEE ALSO *icmpV6, icmpV6Informational, icmpV6MulticastListener, icmpV6NeighborDiscovery, icmpV6OptionLinkLayerDestination, icmpV6OptionLinkLayerSource, icmpV6OptionMaxTransmissionUnit, icmpV6OptionPrefixInformation, icmpV6OptionRedirectedHeader, icmpV6OptionUserDefine, icmpV6UserDefine*

NAME - icmpV6Informational

icmpV6Informational - configures icmpV6 informational messages

SYNOPSIS

icmpV6Informational *sub-command options*

DESCRIPTION

The *icmpV6Informational* command is used to configure icmpV6 informational messages

STANDARD OPTIONS

type	<i>Read-only</i> . The type of ICMPv6 message to be sent. The messageType must first be set in the <i>icmpV6</i> command by calling setType .
code	<i>Read-only</i> . Always 0.
identifier	Identifier for matching Echo Replies and the Echo Request. (<i>default = 0</i>)
sequenceNumber	Sequence number for matching Echo Replies and the Echo Request. (<i>default = 0</i>)
checksum	<i>Read-only</i> . The 16-bit ICMPv6 checksum. This is the ones complement of the ones complement sum of the whole ICMPv6 message, which starts at the message type field. The 'whole' message includes the IPv6 header and extension header fields.

COMMANDS

The **icmpV6Informational** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6Informational **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *icmpV6*

SEE ALSO

icmpV6, *icmpV6Error*, *icmpV6MulticastListener*, *icmpV6NeighborDiscovery*, *icmpV6OptionLinkLayerDestination*, *icmpV6OptionLinkLayerSource*, *icmpV6OptionMaxTransmissionUnit*, *icmpV6OptionPrefixInformation*, *icmpV6OptionRedirectedHeader*, *icmpV6OptionUserDefine*, *icmpV6UserDefine*

NAME - icmpV6MulticastListener

icmpV6MulticastListener - configure icmpV6 multicast listener messages

SYNOPSIS

icmpV6MulticastListener *sub-command options*

DESCRIPTION

The *icmpV6MulticastListener* command is used to configure icmpV6 multicast listener messages

STANDARD OPTIONS

type	<i>Read-only</i> The type of ICMPv6 message to be sent. The messageType must first be set in the <i>icmpV6</i> command by calling setType .
code	<i>Read-only</i> . Always 0.
maximumResponse Delay	(In milliseconds) The maximum delay allowed before a responding Multicast Listener Report message must be sent. (Set by the sender.) If set to '0' it is ignored by receiver. (<i>default = 0</i>)
multicastAddress	For general query type—set to '0'. (<i>default = 0:0:0:0:0:0:0:0</i>) For Multicast-Address-Specific Query—specify an IPv6 multicast address.

COMMANDS

The **icmpV6MulticastListener** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6MulticastListener **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *icmpV6*

SEE ALSO

icmpV6, icmpV6Error, icmpV6Informational, icmpV6NeighborDiscovery, icmpV6OptionLinkLayerDestination, icmpV6OptionLinkLayerSource, icmpV6OptionMaxTransmissionUnit, icmpV6OptionPrefixInformation, icmpV6OptionRedirectedHeader, icmpV6OptionUserDefine, icmpV6UserDefine

NAME - icmpV6NeighborDiscovery

icmpV6neighborDiscovery - configure icmpV6 neighbor discovery messages

SYNOPSIS	icmpV6neighborDiscovery <i>sub-command options</i>
DESCRIPTION	The <i>icmpV6neighborDiscovery</i> command is used to configure icmpV6 neighbor discovery messages
STANDARD OPTIONS	
type	<i>Read-only</i> The type of ICMPv6 message to be sent. The messageType must first be set in the <i>icmpV6</i> command by calling setType . (<i>default = 133</i>)
code	<i>Read-only</i> . Always 0.
checksum	<i>Read-only</i> . The 16-bit ICMPv6 checksum. This is the ones complement of the ones complement sum of the whole ICMPv6 message, which starts at the message type field. The 'whole' message includes the IPv6 header and extension header fields.
currentHopLimit	(for <i>RouterAdvertisementMessage</i>) Default value for the IP Header Hop Count field for outbound IP packets.
destAddress	(for <i>Redirect Message</i>) This is the IPv6 address of the destination. If the destination is a neighbor, this address is also used as the Target address.
enableManagedAddressConfig <i>true/false</i>	(for <i>RouterAdvertisementMessage</i>) If true, hosts use the stateful (administered) protocol for auto-configuration of addresses.
enableOtherStatefulConfig <i>true/false</i>	(for <i>RouterAdvertisementMessage</i>) If true, hosts use the stateful (administered) protocol for auto-configuration of non-addressing (other) information.
enableRouter <i>true/false</i>	(for <i>NeighborAdvertisementMessage</i>) If true, this sender is a router (not a host). (<i>default = false</i>)
enableSolicited <i>true/false</i>	(for <i>NeighborAdvertisementMessage</i>) If true, this neighbor advertisement is sent in response to a neighbor solicitation message. (<i>default = false</i>)
enableOverride <i>true/false</i>	(for <i>NeighborAdvertisementMessage</i>) If true, the information in this advertisement should override the existing entry and update the link layer address. Not for use with anycast addresses. (<i>default = false</i>)
reachableTime	(for <i>RouterAdvertisementMessage</i>) (In milliseconds) Amount of time that a neighbor is assumed to be reachable, following a confirmation of reachable.
retransTimer	(for <i>RouterAdvertisementMessage</i>) (In milliseconds) Time interval between Neighbor Solicitation messages.

routerLifetime (for *RouterAdvertisementMessage*) Default router lifetime, in seconds. If Router Lifetime = 0, this is NOT a default router.

targetAddress (for *NeighborAdvertisement*, *NeighborSolicitation*, or *Redirect Message*) The IPv6 address of the neighbor (target) to which the solicitation was sent. (MUST NOT be multicast IPv6 address.)

For *NeighborAdvertisement* message:

- For solicited advertisements: It is the target address in the Neighbor Solicitation Message.
- For unsolicited advertisements: It is the address with a link-layer address which has changed.

For *Redirect* message:

- This is the same address as the Destination address, if the destination is a neighbor.
- If the target is not a neighbor, this is the address of a router which is a better first-hop node.

COMMANDS

The **icmpV6neighborDiscovery** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6NeighborDiscovery **setDefault**

Sets to IxTclHal default values for all configuration options.

icmpV6NeighborDiscovery **addOption** *optionType*

Adds specified *optionType* to the option list. Specific errors are:

- Invalid option.

icmpV6NeighborDiscovery **delOption**

Deletes the current option. Specific errors are:

- No option found.

icmpV6NeighborDiscovery **getFirstOption**

Gets the first option from the option list. Specific errors are:

- No option found.

icmpV6NeighborDiscovery **getNextOption**

Gets the next option from the option list. Specific errors are:

- No option found.

icmpV6NeighborDiscovery **clearAllOptions**

Clears all the options.

EXAMPLES

See example under *icmpV6*

SEE ALSO

icmpV6, *icmpV6Error*, *icmpV6Informational*, *icmpV6MulticastListener*,
icmpV6OptionLinkLayerDestination, *icmpV6OptionLinkLayerSource*,

*icmpV6OptionMaxTransmissionUnit, icmpV6OptionPrefixInformation,
icmpV6OptionRedirectedHeader, icmpV6OptionUserDefine, icmpV6UserDefine*

NAME - icmpV6OptionLinkLayerDestination

icmpV6OptionLinkLayerDestination - configures the icmpV6 Link Layer Destination option.

SYNOPSIS

icmpV6OptionLinkLayerDestination *sub-command options*

DESCRIPTION

The *icmpV6OptionLinkLayerDestination* command is used to configure the icmpV6 Link Layer Destination option.

This option can be used in all *Neighbor Discovery messages*:

- icmpV6RouterSolicitationMessage
- icmpV6RouterAdvertisementMessage
- icmpV6NeighborSolicitationMessage
- icmpV6NeighborAdvertisementMessage
- icmpV6RedirectMessage

STANDARD OPTIONS

type

Read-only. The value for this option = 2.

length

Read-only. It is the length of the option, and includes type, length, and address fields. One unit of length = 8 octets. The default value = 1.

A length value = 0 is invalid, and the node MUST silently discard an Neighbor Discovery packet where length = 0.

address

(variable length) The target/destination link-layer address. (*default* = 00 00 00 00 00 00)

COMMANDS

The **icmpV6OptionLinkLayerDestination** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6OptionLinkLayerDestination **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *icmpV6*

SEE ALSO

icmpV6, *icmpV6Error*, *icmpV6Informational*, *icmpV6MulticastListener*,
icmpV6NeighborDiscovery, *icmpV6OptionLinkLayerSource*,
icmpV6OptionMaxTransmissionUnit, *icmpV6OptionPrefixInformation*,
icmpV6OptionRedirectedHeader, *icmpV6OptionUserDefine*, *icmpV6UserDefine*

NAME - **icmpV6OptionLinkLayerSource**

icmpV6OptionLinkLayerSource - configures the icmpV6 Link Layer Source option.

SYNOPSIS

`icmpV6OptionLinkLayerSource sub-command options`

DESCRIPTION

The *icmpV6OptionLinkLayerSource* command is used to configure the icmpV6 Link Layer Source option.

This option can be used in all *Neighbor Discovery messages*:

- `icmpV6RouterSolicitationMessage`
- `icmpV6RouterAdvertisementMessage`
- `icmpV6NeighborSolicitationMessage`
- `icmpV6NeighborAdvertisementMessage`
- `icmpV6RedirectMessage`

STANDARD OPTIONS

type

Read-only. The value for this option = 1.

length

Read-only. It is the length of the option, and includes type, length, and address fields. One unit of length = 8 octets. The default value = 1.

A length value = 0 is invalid, and the node MUST silently discard a Neighbor Discovery packet where length = 0.

address

(variable length) The link layer address of the node which sent the packet.
(*default* = `00 00 00 00 00 00`)

COMMANDS

The **icmpV6OptionLinkLayerSource** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6OptionLinkLayerSource `setDefault`

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *icmpV6*

SEE ALSO

icmpV6, icmpV6Error, icmpV6Informational, icmpV6MulticastListener, icmpV6NeighborDiscovery, icmpV6OptionLinkLayerDestination, icmpV6OptionMaxTransmissionUnit, icmpV6OptionPrefixInformation, icmpV6OptionRedirectedHeader, icmpV6OptionUserDefine, icmpV6UserDefine

NAME - icmpV6OptionMaxTransmissionUnit

icmpV6OptionMaxTransmissionUnit - configures the icmpV6 Max Transmission Unit option.

SYNOPSIS

icmpV6OptionMaxTransmissionUnit *sub-command options*

DESCRIPTION

The *icmpV6OptionMaxTransmissionUnit* command is used to configure the icmpV6 Max Transmission Unit option.

This option can be used in all *Neighbor Discovery messages*:

- icmpV6RouterSolicitationMessage
- icmpV6RouterAdvertisementMessage
- icmpV6NeighborSolicitationMessage
- icmpV6NeighborAdvertisementMessage
- icmpV6RedirectMessage

STANDARD OPTIONS

type

Read-only. The value for this option = 5.

length

The length of the option. One unit of length = 8 octets. (*default* = 1)

A length value = 0 is invalid, and the node MUST silently discard a Neighbor Discovery packet where length = 0.

mtu

(32-bit integer) The recommended value of the Maximum Transmission Unit (MTU) on this link. (*default* = 0)

COMMANDS

The **icmpV6OptionMaxTransmissionUnit** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6OptionMaxTransmissionUnit **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *icmpV6*

SEE ALSO

icmpV6, *icmpV6Error*, *icmpV6Informational*, *icmpV6MulticastListener*,
icmpV6NeighborDiscovery, *icmpV6OptionLinkLayerDestination*,
icmpV6OptionLinkLayerSource, *icmpV6OptionPrefixInformation*,
icmpV6OptionRedirectedHeader, *icmpV6OptionUserDefine*, *icmpV6UserDefine*

NAME - icmpV6OptionPrefixInformation

icmpV6OptionPrefixInformation - configures the icmpV6 Prefix Information option.

SYNOPSIS	icmpV6OptionPrefixInformation <i>sub-command options</i>
DESCRIPTION	<p>The <i>icmpV6OptionPrefixInformation</i> command is used to configure the icmpV6 Prefix Information option.</p> <p>This option can be used in all <i>Neighbor Discovery messages</i>:</p> <ul style="list-style-type: none"> • icmpV6RouterSolicitationMessage • icmpV6RouterAdvertisementMessage • icmpV6NeighborSolicitationMessage • icmpV6NeighborAdvertisementMessage • icmpV6RedirectMessage
<hr/>	
STANDARD OPTIONS	
type	<i>Read-only</i> . The value for this option = 3.
length	<p>The length of the option. One unit of length = 8 octets. (<i>default = 1</i>)</p> <p>A length value = 0 is invalid, and the node MUST silently discard a Neighbor Discovery packet where length = 0.</p>
prefixLength	Configures the prefix length. The number of valid bits in the prefix. (<i>default = 0</i>)
enableLinkFlag <i>true/false</i>	If enabled, this prefix can be used for determining if the prefix is on-link. (<i>default = false</i>)
enableAutonomous AddressConfig <i>true/false</i>	If enabled, this prefix can be used for autonomous address configuration. (<i>default = false</i>)
enableRouterAddress <i>true/false</i>	If enabled, indicates a router. The prefix option should not be sent by a router for a link-local prefix. (<i>default = false</i>)
enableSitePrefix <i>true/false</i>	If enabled, indicates a host/site. The prefix option should be ignored by a host, for a link-local prefix. (<i>default = false</i>)
validLifetime	(32-bit integer) The time, starting from packet transmission, that the prefix is valid—in seconds. (0xffffffff = infinity.) (<i>default = 0</i>)
preferredLifetime	(32-bit integer) The time, starting from packet transmission, that the addresses generated from the prefix are “preferred”—in seconds. (0xffffffff = infinity.) (<i>default = 0</i>)
prefix	Can be an IPv6 address or an IPv6 address prefix. The valid leading bits are specified by the setting in the “prefixLength” field. All following bits MUST be set to zero by the sending node and are ignored upon receipt. (<i>default = '0:0:0:0:0:0:0:0'</i>)

COMMANDS

The **icmpV6OptionPrefixInformation** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6OptionPrefixInformation **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *icmpV6*

SEE ALSO

icmpV6, icmpV6Error, icmpV6Informational, icmpV6MulticastListener,
icmpV6NeighborDiscovery, icmpV6OptionLinkLayerDestination,
icmpV6OptionLinkLayerSource, icmpV6OptionMaxTransmissionUnit,
icmpV6OptionRedirectedHeader, icmpV6OptionUserDefine, icmpV6UserDefine

NAME - *icmpV6OptionRedirectedHeader*

icmpV6OptionRedirectedHeader - configures the icmpV6 Redirected Header option.

SYNOPSIS

icmpV6OptionRedirectedHeader *sub-command options*

DESCRIPTION

The *icmpV6OptionRedirectedHeader* command is used to configure the icmpV6 Redirected Header option.

This option can be used in all *Neighbor Discovery messages*:

- *icmpV6RouterSolicitationMessage*
- *icmpV6RouterAdvertisementMessage*
- *icmpV6NeighborSolicitationMessage*
- *icmpV6NeighborAdvertisementMessage*
- *icmpV6RedirectMessage*

STANDARD OPTIONS

type

Read-only. The value for this option = 4.

length

The length of the option. One unit of length = 8 octets. (*default = 1*)

A length value = 0 is invalid, and the node MUST silently discard a Neighbor Discovery packet where length = 0.

ipHeaderAndData

Some of all of the contents of the original IP packet. It consists of as much of the original packet as can be carried in the Redirect message without going over the maximum allowed 1280 octets (bytes).

COMMANDS

The **icmpV6OptionRedirectedHeader** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6OptionRedirectedHeader setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *icmpV6*

SEE ALSO

icmpV6, icmpV6Error, icmpV6Informational, icmpV6MulticastListener, icmpV6NeighborDiscovery, icmpV6OptionLinkLayerDestination, icmpV6OptionLinkLayerSource, icmpV6OptionMaxTransmissionUnit, icmpV6OptionPrefixInformation, icmpV6OptionUserDefine, icmpV6UserDefine

NAME - icmpV6OptionUserDefine

icmpV6OptionUserDefine - configures the icmpV6 User Define option.

SYNOPSIS

icmpV6OptionUserDefine *sub-command options*

DESCRIPTION

The *icmpV6OptionUserDefine* command is used to configure the icmpV6 User Define option.

This option can be used in all *Neighbor Discovery messages*:

- icmpV6RouterSolicitationMessage
- icmpV6RouterAdvertisementMessage
- icmpV6NeighborSolicitationMessage
- icmpV6NeighborAdvertisementMessage
- icmpV6RedirectMessage

STANDARD OPTIONS

type

Read-only The type of ICMPv6 message to be sent. The messageType must first be set in the *icmpV6* command by calling **setType**. (*default = 133*)

length

The length of the option. One unit of length = 1 octet. (*default = 1*)

A length value = 0 is invalid, and the node MUST silently discard a Neighbor Discovery packet where length = 0.

data

User-defined data field. (*default = '00 00 00 00 00 00'*)

COMMANDS

The **icmpV6UserDefine** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6OptionUserDefine **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *icmpV6*

SEE ALSO

icmpV6, icmpV6Error, icmpV6Informational, icmpV6MulticastListener, icmpV6NeighborDiscovery, icmpV6OptionLinkLayerDestination, icmpV6OptionLinkLayerSource, icmpV6OptionMaxTransmissionUnit, icmpV6OptionPrefixInformation, icmpV6OptionRedirectedHeader, icmpV6UserDefine

NAME - **icmpV6UserDefine**

icmpV6UserDefine - configure a user-defined ipV6 header.

SYNOPSIS

icmpV6UserDefine *sub-command options*

DESCRIPTION

The *icmpV6UserDefine* command is used to configure a user-defined ipV6 header.

STANDARD OPTIONS

type	<i>Read-only</i> The type of ICMPv6 message to be sent. The messageType must first be set in the <i>icmpV6</i> command by calling setType .
code	This parameter configures the code. See icmpV6Informational on page A-277 for appropriate codes.
checksum	<i>Read-only</i> . The 16-bit ICMPv6 checksum. This is the ones complement of the ones complement sum of the whole ICMPv6 message, which starts at the message type field. The 'whole' message includes the IPv6 header and extension header fields.

COMMANDS

The **icmpV6UserDefine** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

icmpV6UserDefine [setDefault](#)

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under [icmpV6](#)

SEE ALSO

icmpV6, *icmpV6Error*, *icmpV6Informational*, *icmpV6MulticastListener*,
icmpV6NeighborDiscovery, *icmpV6OptionLinkLayerDestination*,
icmpV6OptionLinkLayerSource, *icmpV6OptionMaxTransmissionUnit*,
icmpV6OptionPrefixInformation, *icmpV6OptionRedirectedHeader*,
icmpV6OptionUserDefine

NAME - IFRHeader

IFRHeader-sets up IFR Header over Fibre Channel.

SYNOPSIS	IFRHeader <i>sub-command options</i>
DESCRIPTION	The Inter-Fabric Routing Extended Header (IFR_Header) provides the necessary information to support fabric-to-fabric routing.
STANDARD OPTIONS	
expirationTime	If the Expiration Time Valid (ETV) bit is set to one, the Expiration Time (Exp_Time) field is used by Inter-Fabric Routers to enforce frame lifetime requirements across the Inter-Fabric.
destinationFabricId	The Destination Fabric Identifier (DF_ID) field is set as specified in FC-IFR.
routingControl	The R_CTL field is a one-byte field that contains routing bits and information bits to categorize the frame function. This field is set to the value 51h to identify the IFR_Header.
hopCount	The count by which the VFT header packet is forwarded in the stream. If the Hop Count Valid (HCV) bit is set to one, the Hop Count (Hop_Cnt) field specifies the number of hops remaining before the frame is discarded.
sourceFabricId	The Source Fabric Identifier (SF_ID) field is set as specified in FC-IFR.
hopCountValid	If Hop Count field is valid, Hop Count Valid bit is set to one. If Hop Count field is invalid, Hop Count Valid bit is set to zero.
expirationTimeValid	If EXP_Time field is valid, Expiry Time Valid bit is set to one. If EXP_Time field is invalid, Expiry Time Valid bit is set to zero.
priority	Specifies the Quality of Service (QoS) value for the frame. When set to zero, is interpreted to contain management information for the class of service.
version	Specifies the version of the IFR_Header. This field is set to a default value of 00b.
COMMANDS	The IFRHeader command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
IFRHeader setDefault	Returns the default settings.
EXAMPLES	See under <i>fibreChannel</i> .

SEE ALSO*fibreChannel*

NAME - igmp

igmp - configure the IGMP parameters for a port on a card on a chassis

SYNOPSIS

igmp sub-command options

DESCRIPTION

The *igmp* command is used to configure the IGMP-specific information used when building IGMP-type packets. Note that *stream* get must be called before this command's get sub-command.

In the case of an IGMP v.3 membership report, the use of an additional command, *igmpGroupRecord*, is needed to hold the group record component of the message. Group records are built in the *igmpGroupRecord* command and added to this command with the *addGroupRecord* sub-command.

STANDARD OPTIONS

enableS true / false

This option is only used for an IGMP v.3 group membership request (that is, *type = membershipQuery* and *version = igmpVersion3*). It is the suppress router-side processing flag. If set, receiving multicast routers will not send timer updates in the normal manner when a query is received. (*default = false*).

groupIpAddress

IP Multicast group address of the group being joined or left. (*default = 0.0.0.0*)

maxResponseTime

The maximum allowed time before sending a responding report in units of 1/10 second. Values from 0 to 127 are represented exactly, values from 128 to 255 are encoded into a floating point number with three bits of exponent and 4 bits of mantissa. A value higher than 255 is silently forced to 255. (*default = 100*)

mode

Describes how to vary the *groupIpAddress* when *repeatCount* is greater than 1.

Option	Value	Usage
<i>igmpIdle</i>	0	(default)
<i>igmpIncrement</i>	1	
<i>igmpDecrement</i>	2	
<i>igmpContIncrement</i>	3	
<i>igmpContDecrement</i>	4	

qqic

This option is only used for an IGMP v.3 group membership request (that is, *type = membershipQuery* and *version = igmpVersion3*). The querier's query interval code, expressed in second. Values from 0 to 127 are represented exactly, values from 128 to 255 are encoded into a floating point number with three bits of exponent and 4 bits of mantissa. A value higher than 255 is silently forced to 255. (*default = 127*)

qrv

This option is only used for an IGMP v.3 group membership request (that is, *type = membershipQuery* and *version = igmpVersion3*). The querier's robustness value, as a value from 0 to 7. (*default = 0*)

repeatCount

Number of times of IGMP messages to be sent. (*default = 1*)

sourceIpAddressList

This option is only used for an IGMP v.3 group membership request (that is, *type* = *membershipQuery* and *version* = *igmpVersion3*). The list of source addresses for the query. (*default* = {})

type

The type of IGMP message to be sent. Options are:

Option	Value	Usage
<i>membershipQuery</i>	17	General or group specific query messages sent by the DUT
<i>membershipReport1</i>	18	(<i>default</i>) An IGMP version 1 message sent by client to inform the DUT of its interest to join a group
<i>dvmrpMessage</i>	19	Distance-Vector Multicast Routing Protocol message
<i>membershipReport2</i>	22	An IGMP version 2 message sent by client to inform the DUT of its interest to join a group
<i>leaveGroup</i>	23	An IGMP version 2 message sent by client to inform the DUT of its interest to leave a group
<i>membershipReport3</i>	34	An IGMP version 3 message sent by a client to inform the DUT of its interest in joining a group.

validChecksum

If set, this causes a valid header checksum to be generated. If unchecked, then the one's complement of the correct checksum is generated. (*default* = true)

version

The version number of IGMP. Options are:

Option	Value	Usage
<i>igmpVersion1</i>	1	version 1
<i>igmpVersion2</i>	2	(<i>default</i>) version 2
<i>igmpVersion3</i>	3	version 3

COMMANDS

The **igmp** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

igmp addGroupRecord

This sub-command is only used for an IGMP v.3 group membership report (that is, *type* = *membershipReport3* and *version* = *igmpVersion3*). The group record described in *igmpGroupRecord* is added to the list in this command. Specific errors are:

- Invalid parameters in the *igmpGroupRecord* command.

igmp cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **igmp** command.

igmp clearGroupRecords

All of the group records in this command are removed.

igmp config option value

Modify the IGMP configuration options of the port. If no *option* is specified, returns a list describing all of the available IGMP options (see STANDARD OPTIONS) for port.

igmp decode capFrame [chassisID cardID portID]

Decodes a captured frame in the capture buffer and updates TclHal. **igmp cget option** command can be used after decoding to get the option data. Specific errors are:

- No connection to a chassis
- Invalid port number
- The captured frame is not a valid Igmp frame

igmp clearGroupRecords index

This sub-command is only used for an IGMP v.3 group membership report (that is, *type = membershipReport3* and *version = igmpVersion3*). The group record at the position in the list indicated by *index* is deleted; the first member in the list has an *index* of 1.

igmp get chassisID cardID portID

Gets the current IGMP configuration of the port with id *portID* on card *cardID*, chassis *chassisID*. Note that *stream* get must be called before this command's get sub-command. Call this command before calling **igmp cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- *stream* get has not bee called yet

igmp getFirstGroupRecord

This sub-command is only used for an IGMP v.3 group membership report (that is, *type = membershipReport3* and *version = igmpVersion3*). The first group record in the list is accessed; the first member in the list has an *index* of 1. The values are available through the *igmpGroupRecord* command. Specific errors are:

- There are no members in the group record list.

igmp getGroupRecord index

This sub-command is only used for an IGMP v.3 group membership report (that is, *type = membershipReport3* and *version = igmpVersion3*). The group record at the position in the list indicated by *index* is accessed; the first member in the list has an *index* of 1. The values are available through the *igmpGroupRecord* command. Specific errors are:

- The *index* does not correspond to an entry in the list.

igmp getNextGroupRecord

This sub-command is only used for an IGMP v.3 group membership report (that is, *type = membershipReport3* and *version = igmpVersion3*). The next group record in the list is accessed; the values are available through the *igmpGroupRecord* command. Specific errors are:

- There are no more members in the group record list.

igmp set *chassisID cardID portID*

Sets the IGMP configuration of the port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **igmp config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

igmp setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host 400-031561
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1

set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

set portMAC {00 00 00 01 01 01}
set portIP {192.168.18.1}
set portMask {255.255.255.0}

set destMAC {00 00 00 01 01 02}
```

```

set destIP      {192.168.18.2}
set destMask   {255.255.255.0}

port setFactoryDefaults $chassis $card $port
port setDefault

# Stream: 256 packets
stream setDefault
stream config -numFrames          256
stream config -sa                 $portMAC
stream config -da                 $destMAC
stream config -dma                stopStream

# Set up IP
ip setDefault
ip config -ipProtocol            igmp
ip config -sourceIpAddr          $portIP
ip config -sourceIpMask          $portMask
ip config -sourceClass           classC
ip config -destIpAddr            $destIP
ip config -destIpMask            $destMask
ip config -destClass             classC
ip set $chassis $card $port

protocol setDefault
protocol config -name              ipV4
protocol config -ethernetType     ethernetII

igmp setDefault
igmp config -groupIpAddress       {224.0.0.1}
igmp config -type                membershipQuery
igmp set $chassis $card $port

stream set $chassis $card $port 1
port set $chassis $card $port

ixWritePortsToHardware portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

stream, ip, udp

NAME - igmpGroupRecord

igmpGroupRecord - specify a IGMP group record used in an IGMP3 Membership Report

SYNOPSIS	igmpGroupRecord <i>sub-command options</i>																					
DESCRIPTION	The <i>igmpGroupRecord</i> command is used to configure a group record element of an IGMP v.3 group membership report. The remainder of the report's fields are configured in the <i>igmp</i> command.																					
STANDARD OPTIONS																						
multicastAddress	A multicast address for a group that the sender interface belongs to. (<i>default</i> = 0.0.0.0)																					
sourceIpAddressList	A list of IPv4 source addresses for the group. (<i>default</i> = {})																					
type	The type of the record.																					
<table border="1"> <thead> <tr> <th>Option</th><th>Value</th><th>Usage</th></tr> </thead> <tbody> <tr> <td>igmpModelsInclude</td><td>1</td><td>A current-state-record which indicates that the interface has a filter mode of INCLUDE for the specified multicast address. The Source Address fields in this Group Record contain the interface's source list for the multicast address.</td></tr> <tr> <td>igmpModelsExclude</td><td>2</td><td>As in <i>igmpModelsInclude</i>, except that the filter mode is EXCLUDE.</td></tr> <tr> <td>igmpChangeToIncludeMode</td><td>3</td><td>A filter-mode-change record that indicates that the interface has changed to INCLUDE filter mode for the specified multicast address. The Source Address fields in this Group Record contain the interface's new source list for the multicast address.</td></tr> <tr> <td>igmpChangeToExcludeMode</td><td>4</td><td>As in <i>igmpChangeToExcludeMode</i>, except that the filter mode is EXCLUDE.</td></tr> <tr> <td>igmpAllowNewSources</td><td>5</td><td>A source-list-change that indicates that the Source Address fields in this Group Record contain a list of the additional sources that the system wishes to hear from, for packets sent to the multicast address. If the change was to an INCLUDE source list, these are the addresses that were added to the list; otherwise these are the addresses that were deleted from the list.</td></tr> <tr> <td>igmpBlockOldSources</td><td>6</td><td>A source-list-change that indicates that the Source Address fields in this Group Record contain a list of the sources that the system no longer wishes to hear from, for packets sent to the multicast address. If the change was to an INCLUDE source list, these are the addresses that were deleted from the list; otherwise these are the addresses that were added to the list.</td></tr> </tbody> </table>		Option	Value	Usage	igmpModelsInclude	1	A current-state-record which indicates that the interface has a filter mode of INCLUDE for the specified multicast address. The Source Address fields in this Group Record contain the interface's source list for the multicast address.	igmpModelsExclude	2	As in <i>igmpModelsInclude</i> , except that the filter mode is EXCLUDE.	igmpChangeToIncludeMode	3	A filter-mode-change record that indicates that the interface has changed to INCLUDE filter mode for the specified multicast address. The Source Address fields in this Group Record contain the interface's new source list for the multicast address.	igmpChangeToExcludeMode	4	As in <i>igmpChangeToExcludeMode</i> , except that the filter mode is EXCLUDE.	igmpAllowNewSources	5	A source-list-change that indicates that the Source Address fields in this Group Record contain a list of the additional sources that the system wishes to hear from, for packets sent to the multicast address. If the change was to an INCLUDE source list, these are the addresses that were added to the list; otherwise these are the addresses that were deleted from the list.	igmpBlockOldSources	6	A source-list-change that indicates that the Source Address fields in this Group Record contain a list of the sources that the system no longer wishes to hear from, for packets sent to the multicast address. If the change was to an INCLUDE source list, these are the addresses that were deleted from the list; otherwise these are the addresses that were added to the list.
Option	Value	Usage																				
igmpModelsInclude	1	A current-state-record which indicates that the interface has a filter mode of INCLUDE for the specified multicast address. The Source Address fields in this Group Record contain the interface's source list for the multicast address.																				
igmpModelsExclude	2	As in <i>igmpModelsInclude</i> , except that the filter mode is EXCLUDE.																				
igmpChangeToIncludeMode	3	A filter-mode-change record that indicates that the interface has changed to INCLUDE filter mode for the specified multicast address. The Source Address fields in this Group Record contain the interface's new source list for the multicast address.																				
igmpChangeToExcludeMode	4	As in <i>igmpChangeToExcludeMode</i> , except that the filter mode is EXCLUDE.																				
igmpAllowNewSources	5	A source-list-change that indicates that the Source Address fields in this Group Record contain a list of the additional sources that the system wishes to hear from, for packets sent to the multicast address. If the change was to an INCLUDE source list, these are the addresses that were added to the list; otherwise these are the addresses that were deleted from the list.																				
igmpBlockOldSources	6	A source-list-change that indicates that the Source Address fields in this Group Record contain a list of the sources that the system no longer wishes to hear from, for packets sent to the multicast address. If the change was to an INCLUDE source list, these are the addresses that were deleted from the list; otherwise these are the addresses that were added to the list.																				

COMMANDS

The **igmpGroupRecord** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

igmpGroupRecord `cget` *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **igmpGroupRecord** command.

igmpGroupRecord `config` *option value*

Modify the IP address table configuration options of the port. If no *option* is specified, returns a list describing all of the available **igmpGroupRecord** options (see STANDARD OPTIONS) for port.

igmpGroupRecord `setDefault`

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under [igmp](#).

SEE ALSO

[igmp](#)

NAME - *interfaceEntry*

interfaceEntry - configure an interface associated with a port

SYNOPSIS

interfaceEntry sub-command options

DESCRIPTION

The *interfaceEntry* command is used to configure a single interface associated with a port. Interface entries hold one or more IPv4 or IPv6 addresses. Data from this command must be added to the interface table using the *interfaceTable* command.

STANDARD OPTIONS

atmEncapsulation

For ATM type load modules, this is the type of ATM encapsulation that is used on the interface.

Option	Value	Usage
atmEncapsulationVccMuxIPv4Routed	101	
atmEncapsulationVccMuxBridgedEthernetFCS	102	
atmEncapsulationVccMuxBridgedEthernetNoFCS	103	
atmEncapsulationVccMuxIPv6Routed	104	
atmEncapsulationVccMuxMPLSRouted	105	
atmEncapsulationLLCRoutedCLIP	106	
atmEncapsulationLLCBridgedEthernetFCS	107	(default)
atmEncapsulationLLCBridgedEthernetNoFCS	108	
atmEncapsulationLLC PPPoA	109	
atmEncapsulationVccMuxPPPoA	110	

atmVci

For ATM type cards, the VCI associated with the interface. (*default = 0*)

atmVpi

For ATM type cards, the VPI associated with the interface. (*default = 0*)

connectedVia

If *interfaceType* is set to *interfaceTypeRouted*, then this is the description of the interface that this internal interface is made available through. (*default = “”*)

description

An optional description for the interface. This may be used to later access a particular interface by name in the *interfaceTable* command. (*default = “”*)

enable true / false

Enables the use of this interface entry. (*default = false*)

enableDcbx true / false

Enables the use of DCBX negotiation on this interface entry. (*default = false*)

See *enableLldp*, below.

enableDhcp true / false

Enables the use of DHCP negotiation on this interface entry. If this option is *true*, then no address items may be added to this interface entry. Any existing IPv4 addresses are deleted. (*default = false*)

enableDhcpV6 <i>true / false</i>	Enables the use of DHCPv6 negotiation on this interface entry. If this option is <i>true</i> , then no address items may be added to this interface entry. Any existing IPv4 addresses are deleted. (<i>default = false</i>)
enableFlogi <i>true / false</i>	Enable Fabric login (for FCoE protocol). (<i>default = false</i>)
enableGreChecksum true / false	If <i>interfaceType</i> is <i>interfaceTypeGre</i> , this enables the presence of the optional <i>Checksum</i> and <i>Reserved1</i> fields of the GRE header. The <i>Checksum</i> is set to a correct value and the <i>Reserved1</i> field is set to 0. (<i>default = 0.0.0.0</i>)
enableGreKey true / false	If <i>true</i> , the <i>Key</i> field is included in outgoing packets using the value in the <i>greOutKey</i> field. (<i>default = false</i>)
enableGreSequence true / false	If <i>true</i> , the <i>Sequence Number</i> field is included in outgoing packets. (<i>default = false</i>)
enableLldp <i>true / false</i>	Enables the use of LLDP Tx and Rx negotiation on this interface entry. (<i>default = false</i>)
	Note: Since DCBX is an acknowledged protocol which uses LLDP, for the protocol to operate correctly, both LLDP Rx and Tx are enabled on the interface on which DCBX runs.
enablePtp <i>true / false</i>	Enables the use of PTP on this interface. When set to true, the PTP configuration is stored in the <i>ptpProperties</i> command. (<i>default = false</i>)
enableVlan <i>true / false</i>	Enables the use of the VLAN on this interface. (<i>default = false</i>)
eui64Id	The EUI-64 ID associated with POS boards with IPv6 support. (<i>default = {00 00 00 FF FE 00 00 00}</i>)
greDestIpAddress	If <i>interfaceType</i> is <i>interfaceTypeGre</i> , this is the destination IP address to be set in the GRE header. (<i>default = 0.0.0.0</i>)
greInKey	If <i>interfaceType</i> is <i>interfaceTypeGre</i> , this is the key used to match incoming packets. (<i>default = 0</i>)
greOutKey	If <i>interfaceType</i> is <i>interfaceTypeGre</i> and <i>enableGreKey</i> is set to <i>true</i> , this is the key inserted in outgoing packets. (<i>default = 0</i>)
greSourceIpAddress	If <i>interfaceType</i> is <i>interfaceTypeGre</i> , this is the source IP address to be set in the GRE header. (<i>default = 0.0.0.0</i>)
interfaceType	The type of interface being defined.

Option	Value	Usage
<i>interfaceTypeConnected</i>	0	(<i>default</i>) A standard, connected interface
<i>interfaceTypeGre</i>	4	A GRE internal interface. The <i>connected-Via</i> option must be set to the name of an interface of type <i>interfaceTypeConnected</i> .
<i>interfaceTypeRouted</i>	5	An internal, unconnected interface. The <i>connectedVia</i> option must have the name of a connected interface that this interface is routed through.
<i>interfaceTypeNpiv</i>	6	An NPIV type interface.

Option	Value	Usage
<i>interfaceTypePtp</i>	7	A PTP type interface. (Note: When enablePTP is set 'true' in the interfaceEntry command, the PTP configuration is stored in the ptpproperties command.)

ipv6Gateway

There can be one gateway per IPv6 interface (*default = '0:0:0:0:0:0:0:0'*)

macAddress

The MAC address of the interface. (*default = "00 00 00 00 00 00"*)

mtu

Sets the Maximum Transmission Unit size, in kilobytes. The range possible depends on the port type. (*default = 1500*)

vcatCircuit

Sets the circuit ID for an IxRouter interface. (*default = ""*)

vlanId

If *enableVlan* is *true*, the routing protocols are VLAN encapsulated with this ID. Although a value of '0' is allowed, VLAN IDs normally start at 1. (*default = 0*)

vlanPriority

If *enableVlan* is *true*, the user priority of the VLAN ID tag (from 0 to 7). (*default = 0*)

vlanTPID

If *enableVlan* is *true*, the VLAN Tag Protocol ID. EtherTypes identify the protocol that follows the VLAN header. (*default = 0x8100*)

DEPRECATED OPTIONS
atmMode

The *encapsulation* associated with the *atmHeader* is used instead.

COMMANDS

The **interfaceEntry** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

interfaceEntry addItem *ipType*

Adds an IPv4 or IPv6 address, depending on the value of *ipType*, which is one of these:

Option	Value	Usage
<i>addressTypeIpV4</i>	17	An IPv4 address is added from the options associated with the <i>interfaceIpV4</i> command.
<i>addressTypeIpV6</i>	18	An IPv6 address is added from the options associated with the <i>interfaceIpV6</i> command.

Only one IPv4 address can be associated with an interface at this time.

Specific errors are:

- Invalid address configuration.

interfaceEntry clearAllItems *ipType*

Clears all IPv4 and IPv6 addresses of the interface, depending on the value of *ipType*, which is one of these:

Option	Value	Usage
<i>addressTypeIpV4</i>	17	An IPv4 address is added from the options associated with the <i>interfaceIpV4</i> command.
<i>addressTypeIpV6</i>	18	An IPv6 address is added from the options associated with the <i>interfaceIpV6</i> command.

interfaceEntry delItem *ipType* [*ipAddr*]

Removes an address of type *ipType* (see the *addItem* sub-command above for a description of the items). The address may either be specified with the *ipAddr* of the entry or the current interface as accessed with *getFirstItem*, *getNextItem* and *getItem*. Separate current list pointers are kept for IPv4 and IPv6 items. Specific errors are:

- There is no object with this ID.

interfaceEntry getFirstItem *ipType*

Gets the first address of type *ipType* (see the *addItem* sub-command above for a description of the items) from the interface entry. Separate current list pointers are kept for IPv4 and IPv6 items. The data may be accessed with the *interfaceIpV4* or *interfaceIpV6* command. Specific errors are:

- Required commands have not been called.
- The list is empty.

interfaceEntry getItem *ipAddress*

Gets the IPv4 or IPv6 item from the interface entry which matches the specified *ipAddress*. The type of entry is figured out from the format of the *ipAddress*. The data may be accessed with the *interfaceIpV4* or *interfaceIpV6* command. Specific errors are:

- Required commands have not been called.
- There is no object with this ID.

interfaceEntry getNextItem *ipType*

Gets the next interface entry from the interface table. The data may be accessed with the *interfaceEntry* command. Specific errors are:

- Required commands have not been called.
- There are no more objects in the list.

interfaceEntry setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *interfaceTable*.

SEE ALSO

interfaceTable, *dhcpV4DiscoveredInfo*, *dhcpV4Properties*, *dhcpV4Tlv*

NAME - interfaceIpV4

interfaceIpV4 - configure an IPv4 address for inclusion in an interface entry

SYNOPSIS *interfaceIpV4 sub-command options*

DESCRIPTION The *interfaceIpV4* command is used to configure the IPv4 address specific information used when building an interface table. An *interfaceIpV4* is added to an interface entry using the *interfaceEntry* command.

STANDARD OPTIONS

gatewayIpAddress	The gateway IP address. (<i>default</i> = 0.0.0.0)
ipAddress	The IPv4 address. (<i>default</i> = 0.0.0.0)
maskWidth	The network mask associated with the address. Valid values: 1-30. (<i>default</i> = 24)

COMMANDS The **interfaceIpV4** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

interfaceIpV4 cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **interfaceIpV4** command.

interfaceIpV4 config option value

Modify the IP address table configuration options of the port. If no *option* is specified, returns a list describing all of the available **interfaceIpV4** options (see STANDARD OPTIONS) for port.

interfaceIpV4 setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES See examples under *interfaceTable*.

SEE ALSO

NAME - interfaceIpV6

interfaceIpV6 - configure an IPv6 address for inclusion in an interface entry

SYNOPSIS

interfaceIpV6 *sub-command options*

DESCRIPTION

The *interfaceIpV6* command is used to configure the IPv6 address specific information used when building an interface table. An *interfaceIpV6* is added to an interface entry using the *interfaceEntry* command.

STANDARD OPTIONS

ipAddress The IPv6 address. (*default = "0:0:0:0:0:0:0:0"*)

maskWidth The network mask associated with the address. (*default = 64*)

COMMANDS

The **interfaceIpV6** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

interfaceIpV6 **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **interfaceIpV6** command.

interfaceIpV6 **config** *option value*

Modify the IP address table configuration options of the port. If no *option* is specified, returns a list describing all of the available **interfaceIpV6** options (see STANDARD OPTIONS) for port.

interfaceIpV6 **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *interfaceTable*.

SEE ALSO

NAME - interfaceTable

interfaceTable - configure the interfaces associated with a port

SYNOPSIS	interfaceTable <i>sub-command options</i>
DESCRIPTION	<p>The <i>interfaceTable</i> command is used to configure interfaces associated with a port. Interfaces hold <i>interfaceEntry</i> elements, each of which includes multiple IPv4 and IPv6 addresses. Note that the <i>select</i> command must be used before any other sub-commands to indicate the chassis, card and port in use.</p> <p>For IPv4, DHCPv4 or DHCPv6 may be enabled on an interface by interface basis in <i>interfaceEntry</i>. DHCP parameters are set <i>dhcpV4Properties</i> and <i>dhcpV6Properties</i> commands at the time that <i>interfaceTable addInterface</i> is called. They are retrieved when the <i>get*Interface</i> sub-commands are called. The address and other parameters assigned from the DHCP server may be retrieved from the port by using <i>requestDiscoveredTable</i> followed by <i>getDhcpV4DiscoveredInfo</i>.</p> <p>Similarly, when using IPv6, addresses for the interfaces and neighbor addresses are automatically discovered and are available by calling <i>sendRouterSolicitation</i>, <i>requestDiscoveredTable</i> and <i>getDiscoveredList</i>.</p> <p>Note: If more than a few DHCP interfaces are being defined, it is important that you wait until they are fully defined by monitoring the <i>dhcpV4EnabledInterfaces</i> and <i>dhcpV6EnabledInterfaces</i> statistic in the <i>stat</i> command. Likewise, the DHCP server may require some amount of time to answer all DHCP server requests. You can test for its completion by calling <i>interfaceTable requestDiscoveredTable</i> and then monitoring the <i>dhcpV4AddressesLearned</i> and <i>dhcpV6AddressesLearned</i> statistic in the <i>stat</i> command. This requires that <i>enableDhcpStats</i> be <i>true</i> in the <i>stat</i> command.</p>
<hr/>	
STANDARD OPTIONS	
dhcpV4RequestRate	The user-specified maximum number of Request messages that can be sent per second from the client to the server, requesting an IPv4 address. A value of zero (0) indicates that there is no rate control, that is, requests are sent as fast as possible.
dhcpV6RequestRate	The user-specified maximum number of Request messages that can be sent per second from the client to the server, requesting an IPv6 address. A value of zero (0) indicates that there is no rate control, that is, requests are sent as fast as possible.
dhcpV4Maximum Out-standingRequests	The maximum number of DHCP V4 requests that can be pending, waiting replies from the server. If this number is reached, no further requests can be sent until an acknowledgment is received for a pending request.
dhcpV6Maximum Out-standingRequests	The maximum number of DHCP V6 requests that can be pending, waiting replies from the server. If this number is reached, no further requests can be sent until an acknowledgment is received for a pending request.

enableFcfcMac	Enables FCF MAC address.
enablePMacInFpma	Enables PMAC.
enableNameIdInVLAN Discovery	Enables Name ID parameter in Discovery VLAN.
enableTargetLinkLayer AddrOption	Enables Target Link Layer Address option.
enableAutoNeighbor Discovery	Enables Auto Neighbor Discovery parameter. If true and then MAC interface is enabled, the Discovered Neighbors parameters are automatically available.
enableAutoArp	Enables Auto ARP option. If true and then MAC interface is enabled, the Learned IP Addresses and Learned MAC Addresses are automatically available.
fcfMacCollectionTime	The FCF MAC collection time.
fcoeNumRetries	FCoE number of retries before being marked as Failure. (<i>default = 5</i>)
fcoeRetryInterval	FCoE interval between retries. (<i>default = 2000</i>)
fcoeRequestRate	FCoE maximum rate (packets/second). (<i>default = 500</i>)
fipVersion	FIP version. (<i>default = 1</i>)

Option	Value	Usage
<i>fipVersion0</i>	0	The version in incoming packets should have the same value as the one configured in IxExplorer, otherwise packets is dropped.
<i>fipVersion1</i>	1	(<i>default</i>) See Usage for <i>fipVersion0</i> , above.
<i>fipVersionAuto</i>	8888	The protocol sends packets matching the fipversion of incoming packets. If incoming packets are version 0, then FIP sends version 0 packets; if incoming packets are version 1 then FIP sends version 1 packets.

COMMANDS

The **interfaceTable** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

interfaceTable addInterface [*type*]

Adds the interface specified in the *interfaceEntry* command. The *type* options should be one of these:

Option	Value	Usage
<i>interfaceTypeConnected</i>	0	(<i>default</i>) A standard, connected interface
<i>interfaceTypeGre</i>	4	A GRE internal interface. The <i>connected-Via</i> option must be set to the name of an interface of type <i>interfaceTypeConnected</i> .

Option	Value	Usage
<i>interfaceTypeRouted</i>	5	An internal, unconnected interface. The <i>connectedVia</i> option must have the name of a connected interface that this interface is routed through.
<i>interfaceTypeNpiv</i>	6	An NPIV type interface.
<i>interfaceTypePtp</i>	7	A PTP type interface. (Note: When <i>enablePTP</i> is set 'true' in the <i>interfaceEntry</i> command, the PTP configuration is stored in the <i>ptpProperties</i> command.)

Specific errors are:

- The *select* sub-command has not been called successfully before

interfaceTable clearAllInterfaces [type]

Clears all of the interfaces associated with the port selected in *interfaceTable select*. If specified, only the interfaces defined with the *interfaceEntry*'s *interfaceType* equal to *type* are cleared. Specific errors are:

- The *select* sub-command has not been called successfully before

interfaceTable clearDiscoveredNeighborTable

Clears all of the discovered neighbors associated with the port selected in *interfaceTable select*:

- The *select* sub-command has not been called successfully before

interfaceTable clearPtpHistogramData description

Clears all of the accumulated PTP histogram data associated with the selected interface. This command also stops the collection process. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- Invalid port.
- Invalid *description*.
- There is no interface with this *description*.

interfaceTable config option value

Modify the **interfaceTable** configuration options of the port. If no *option* is specified, returns a list describing all of the available **interfaceTable** options (see STANDARD OPTIONS).

interfaceTable cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **interfaceTable** command.

interfaceTable delInterface [description]

Removes an interface. The interface may either be specified with the *description* given when the interface was added with *addInterface* or the current interface as accessed with *getFirstInterface*, *getNextInterface* and *getInterface*. Specific errors are:

- The *select* sub-command has not been called successfully before

interfaceTable getDcbxDiscoveredInfo *description*

Gets the DCBX interface description and other information from the interface table which matches the specified *description*. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- Invalid port.
- Invalid *description*.
- There is no interface with this *description*.
- There is no discovered information for the interface.

interfaceTable getDhcpV4DiscoveredInfo *description*

Gets the DHCP assigned address and other information from the interface table which matches the specified *description*. The data may be accessed with the *dhcpV4DiscoveredInfo* command. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- Invalid port.
- Invalid *description*.
- There is no interface with this *description*.
- There is no discovered information for the interface.

interfaceTable getDhcpV6DiscoveredInfo *description*

Gets the DHCPv6 assigned address and other information from the interface table which matches the specified *description*. The data may be accessed with the *dhcpV6DiscoveredInfo* command. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- Invalid port.
- Invalid *description*.
- There is no interface with this *description*.
- There is no discovered information for the interface.

interfaceTable getDiscoveredList [*description*]

Obtains the discovered neighbor and address list corresponding to an interface. The interface may either be specified with the *description* given when the interface was added with *addInterface* or the current interface as accessed with *getFirstInterface*, *getNextInterface* and *getInterface*. This command should be called until it returns TCL_OK (0), at which time the list has been retrieved. An additional delay may be necessary if there are more than a few entries expected. The data may be accessed with the *discoveredList* command.

interfaceTable getFcoeDiscoveredInfo [*description*]

Gets the FCoE assigned address and other information from the interface table which matches the specified *description*. The data may be accessed with the *fcoeDiscoveredInfo* command. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- Invalid port.
- Invalid *description*.
- There is no interface with this *description*.
- There is no discovered information for the interface.

interfaceTable getFirstInterface [*type*]

Gets the first interface entry from the interface table. The data may be accessed with the *interfaceEntry* command. If specified, only the interfaces defined with the *interfaceEntry*'s *interfaceType* equal to *type* are accessed. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- The list is empty.
- No entry of the *type* specified exists.

interfaceTable getInterface *description*

Gets the interface entry from the interface table which matches the specified *description*. The data may be accessed with the *interfaceEntry* command. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- There is no object with this ID.

interfaceTable getNextInterface [*type*]

Gets the next interface entry from the interface table. The data may be accessed with the *interfaceEntry* command. If specified, only the interfaces defined with the *interfaceEntry*'s *interfaceType* equal to *type* are accessed. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- There are no more objects in the list.
- No more entries of the *type* specified exists.

interfaceTable getPtpDiscoveredInfo [*description*]

Gets the PTP assigned address and other information from the interface table which matches the specified *description*. The data may be accessed with the *ptpDiscoveredInfo* command. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- Invalid port.
- Invalid *description*.
- There is no interface with this *description*.
- There is no discovered information for the interface.

interfaceTable ping [*description*][*ipType*][*ipAddress*]

Sends a ping to the specified IPv4 and/or IPv6 destination. Must be enabled in *protocolServer* to work. The available ipType are *addressTypeIpV4* and *addressTypeIpV6*.

Specific errors are:

- The interface is not enabled for the port.
- A port has not been selected by the *interfaceTable select* command.
- A network problem has occurred.
- Ping not enabled in *protocolServer*.
- Invalid IP type.
- Invalid IP address.
- Invalid interface description.

interfaceTable requestDiscoveredTable

Requests that the IPv6 discovered neighbors and both IPv6 and IPv4-DHCP interfaces addresses be sent back from the hardware. This should be followed by use of the *getDiscoveredList* command when used with IPv6 discovered neighbors. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- A network problem has occurred.

interfaceTable savePtpHistogramData *description filePath*

Saves to disk all the accumulated PTP histogram data associated with the selected interface. The save file is of the type comma-separated-values (.csv). Note that for the *savePtpHistogramData* method there is no enforcement of the file name. You may specify it as desired. IxExplorer suggests the following filename format as a convenience:

PTPHistogram-<PTPClockId>_<PTPPortId>.csv

Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- Invalid port.
- Invalid *description*.
- There is no interface with this *description*.
- Invalid *filePath*

interfaceTable select *chassisID cardID portID*

Accesses the interface table for the indicated port. Specific errors are:

- No connection to the chassis
- Invalid port specified

interfaceTable sendArp [*description*]

Sends an ARP request corresponding to an interface or all enabled interfaces. The interface may either be specified with the *description* given when the interface was added with *addInterface* or, if omitted, all enabled interfaces are ARP'd. You must use the *requestDiscoveredTable* command before using this command. This should be followed by a call to the *requestDiscoveredList* command after which point the data may be accessed with the *getDiscoveredList* command. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- A network problem has occurred.

interfaceTable sendArpClear

Clears the ARP table for all enabled interfaces. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- A network problem has occurred.

interfaceTable sendArpRefresh [*description*]

Rereads the ARP table corresponding to an interface or all enabled interfaces from the port's CPU. The interface may either be specified with the *description* given when the interface was added with *addInterface* or, if omitted, all enabled interfaces are queried. This should be followed by a call to the

requestDiscoveredList command after which point the data may be accessed with the *getDiscoveredList* command. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- A network problem has occurred.

interfaceTable sendNeighborClear

Sends a neighbor clear message that clears the neighbor cache for all the enabled interfaces for the port. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- A network problem has occurred.

interfaceTable sendNeighborRefresh

Sends a refresh message that allows a device to refresh a neighbor that exists and is reachable. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- A network problem has occurred.

interfaceTable sendNeighborSolicitation

Allows a device to check that a neighbor exists and is reachable, and to initiate address resolution. The Neighbor Advertisement message confirms the existence of a host or router, and also provides Layer 2 address information when needed. This request corresponds to all multicast enabled interfaces. The interface may either be specified with the description that was given when the interface was added with *addInterface*, or, if omitted, all enabled interfaces are sent Neighbor Solicitation /message. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- A network problem has occurred.

interfaceTable sendRouterSolicitation [description]

Sends a router solicitation packet (IPv6) corresponding to an interface. The interface may either be specified with the *description* given when the interface was added with *addInterface* or the current interface as accessed with *getFirstInterface*, *getNextInterface* and *getInterface*. This should be followed by a call to the *requestDiscoveredList* command after which point the data may be accessed with the *getDiscoveredList* command. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- A network problem has occurred.

interfaceTable setInterface [description]

Sets an interface entry in the interface to the specified *description*. The data may be accessed with the *interfaceEntry* command. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- There is no interface with that description.

interfaceTable startPtpHistogramData description

Starts (or resumes) collecting the PTP histogram data associated with the selected interface. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.

- Invalid port.
- Invalid *description*.
- There is no interface with this *description*.

interfaceTable **stopPtpHistogramData** *description*

Stops collecting the PTP histogram data associated with the selected interface. Stopping the collection of data does not cause any accumulated data to be lost. Specific errors are:

- A port has not been selected by the *interfaceTable select* command.
- Invalid port.
- Invalid *description*.
- There is no interface with this *description*.

interfaceTable **write**

Sends any changes made to the interface table to the protocol server. If more than a few interfaces are being defined, it is important that you wait until they are fully defined by monitoring the *dhcpV4EnabledInterfaces* and *dhcpV6EnabledInterfaces* statistic in the *stat* command. This requires that *enableDhcpStats* be *true* in the *stat* command. Possible errors include:

- A port has not been selected by the *interfaceTable select* command.
- A network problem has occurred.

EXAMPLES

Example 1

```

package req IxTclHal
ixConnectToChassis loopback

set chassisID [chassis cget -id]
set cardID 1
set portID 3
set pingAddress 1.1.1.2

set portList [list [list $chassisID $cardID $portID]]

protocolServer get $chassisID $cardID $portID
protocolServer config -enablePingResponse $::true
protocolServer set $chassisID $cardID $portID
protocolServer write $chassisID $cardID $portID

interfaceTable select $chassisID $cardID $portID

# Get the interface description from the first interface that
# happens to be our port
interfaceTable getFirstInterface
set desc [interfaceEntry cget -description]

# Available ipType are addressTypeIpV4 and addressTypeIpV6
# interfaceTable $description ipType $ipAddress

# Clear the stats in order to see if you received ping correctly
ixClearStats portList

# Send ping request
interfaceTable ping $desc addressTypeIpV4 $pingAddress

```

```

# Wait for ping reply to come back
after 2000

ixRequestStats portList
statList get $chassisID $cardID $portID

puts "\n *** Ping request sent on $chassisID $cardID $portID:
[statList cget -txPingRequest]"
puts "****Ping reply received on $chassisID $cardID $portID:
[statList cget -rxPingReply]"

```

Example 2

```

package req IxTclHal

# Define parameters used by OSPF router
set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set ch [ixGetChassisID $host]

# Port is: card 4, port 1
set ca 4
set po 1
set portList [list [list $ch $ca $po]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

set myMac          {00 0a de 01 01 01}
set myMac2         {00 0a de 01 01 02}
set router         101.101.9.2
set router2        101.101.10.2
set neighbor       101.101.9.1
set interfaceIpMask 255.255.255.0

# Set up the interface table for IPv4 and IPv6 interfaces

```

```

# on the port
interfaceTable select $ch $ca $po
interfaceTable clearAllInterfaces

interfaceIpV6 setDefault
interfaceIpV6 config -ipAddress {0:0:0:0:0:0:0:1}
interfaceIpV6 config -maskWidth 64
interfaceEntry addItem addressTypeIpV6

interfaceIpV4 setDefault
interfaceIpV4 config -ipAddress $router
interfaceIpV4 config -gatewayIpAddress $neighbor
interfaceIpV4 config -maskWidth 24
interfaceEntry addItem addressTypeIpV4

interfaceEntry setDefault
interfaceEntry config -enable true
interfaceEntry config -description {Port 04:01 Interface-1}
interfaceEntry config -macAddress $myMac
interfaceEntry config -ipV6Gateway (1:1:1:1:1:0:0:55)

interfaceTable addInterface

interfaceEntry clearAllItems addressTypeIpV4
interfaceEntry clearAllItems addressTypeIpV6

interfaceIpV6 setDefault
interfaceIpV6 config -ipAddress {0:0:0:0:0:0:0:2}
interfaceIpV6 config -maskWidth 64
interfaceEntry addItem addressTypeIpV6

interfaceIpV4 setDefault
interfaceIpV4 config -ipAddress $router2
interfaceIpV4 config -gatewayIpAddress $neighbor
interfaceIpV4 config -maskWidth 24
interfaceEntry addItem addressTypeIpV4

interfaceEntry setDefault
interfaceEntry config -enable true
interfaceEntry config -description {Port 04:01 Interface-2}
interfaceEntry config -macAddress $myMac2
interfaceTable addInterface
interfaceTable write

# Now go through the table and print all interfaces and addresses
interfaceTable select $ch $ca $po

# Loop through all interfaces
for {set bRes [interfaceTable getFirstInterface]} \
    {$bRes == 0} {set bRes [interfaceTable getNextInterface]} {

    ixPuts "Interface: " [interfaceEntry cget -description] \
        ", MAC: " \
        [interfaceEntry cget -macAddress]

    # Get the one optional IpV4 entry
    if {[interfaceEntry getFirstItem addressTypeIpV4] == 0} {
        ixPuts "\tIPv4 Address:"
        ixPuts "\t\t" [interfaceIpV4 cget -ipAddress] "/" \
            [interfaceIpV4 cget -maskWidth] ", GW: " \
            [interfaceIpV4 cget -gatewayIpAddress]
    }
}

```

```

}

# Loop through all IPv6 addresses
for {set bRes [interfaceEntry getFirstItem addressTypeIpV6]}\ \
    {$bRes == 0} \
    {set bRes [interfaceEntry getNextItem addressTypeIpV6]} {
    ixPuts "\tIPv6 Addresses:"
    ixPuts "\t\t [interfaceIpV6 cget -ipAddress] "/" \
        [interfaceIpV6 cget -maskWidth]
}
}

# Now request and get the discovered neighbor and address list
# for all interfaces

interfaceTable clearDiscoveredNeighborTable

# Loop through all interfaces
for {set bRes [interfaceTable getFirstInterface]}\ \
    {$bRes == 0} {set bRes [interfaceTable getNextInterface]} {

    ixPuts "Interface: " [interfaceEntry cget -description]

    # Send a request command on the network
    if [interfaceTable sendRouterSolicitation] {
        ixPuts "Can't send router solicitation"
    } else {

        # Wait for responses
        after 5000
        # Ask for the discovered table
        if [interfaceTable requestDiscoveredTable] {
            ixPuts "Can't request discovered table"
        } else {

            # Now wait until it finishes reading back
            ixPuts -nonewline "Waiting."
            for {set count 0} \
                {[interfaceTable getDiscoveredList] != 0 && \
                    $count < 10} {incr count} {
                ixPuts -nonewline "."
                after 1000
            }
            ixPuts ""
            if {$count == 10} {
                ixPuts "Can't get discovered list"
            } else {

                # Wait for a bit to ensure that all of the entries
                # have been retrieved.
                after 5000
                # Get the discovered addresses
                for {set bRes [discoveredList getFirstAddress]}\ \
                    {$bRes == 0} \
                    {set bRes [discoveredList getNextAddress]} {
                        ixPuts "\tDiscovered address: " \
                            [discoveredAddress cget -ipAddress]
                }
                # Get the discovered neighbors
                for {set bRes [discoveredList getFirstNeighbor]}\ \
                    {$bRes == 0} \
                    {set bRes \
                        [discoveredList getNextNeighbor]} {

```

```

        for {set bRes \
            [discoveredNeighbor getFirstAddress] } \
            {$bRes == 0} \
            {set bRes \
                [discoveredNeighbor getNextAddress] } {
            ixPuts -none newline \
                "\tDiscovered neighbor: address: " \
                [discoveredNeighbor cget -ipRouter]
            ixPuts ", mac address: " \
                [discoveredNeighbor cget -macAddress]
        }
    }
}
}

# Send ARP to each interface one at a time
if {[interfaceTable select $ch $ca $po]} {
    logMsg "Error selecting port"
    set retCode "FAIL"
}

for {set i 1} {$i < 2} {incr i} {
    if {[interfaceTable sendArp "Port 04:01 Interface-$i"]} {
        logMsg "Error sending Arp for interface $i"
        set retCode "FAIL"
    }
}

if {[interfaceTable requestDiscoveredTable]} {
    logMsg "Error in requesting discoveredTable"
    set retCode "FAIL"
}
after 10000

for {set i 1} {$i < 2} {incr i} {
    if {! [interfaceTable getDiscoveredList \
        "Port 04:01 Interface-$i"]} {
        # Use discoveredList as above
    } else {
        set retCode "FAIL"
    }
}

#####
#####
#
#   DHCP example
#
#####
#####

# Init the interface table
if [interfaceTable select $ch $ca $po] {
    logMsg "Error selecting $ch $ca $po"
    set retCode "FAIL"
}
interfaceTable clearAllInterfaces

# Initialize and set DHCP properties for interface
dhcpV4Properties setDefault

```

```

dhcpV4Properties removeAllTlvs
dhcpV4Properties config -clientId           Client1
dhcpV4Properties config -serverId          1.1.1.2
dhcpV4Properties config -vendorId          Ixial
v4Properties config -renewTimer            600

# Define a type 2 TLV
dhcpV4Tlv      setDefault
dhcpV4Tlv      config -type                2
dhcpV4Tlv      config -value              {AA AB 22}
if [dhcpV4Properties addTlv] {
    logMsg "Error in dhcpV4Properties addTlv"
    set retCode "FAIL"
}

# Define a type 12 TLV
dhcpV4Tlv      config -type               12
dhcpV4Tlv      config -value              {A1 A2 B1 B2}
if [dhcpV4Properties addTlv] {
    logMsg "Error in dhcpV4Properties addTlv"
    set retCode "FAIL"
}

# Define an interface entry that uses DHCP
interfaceEntry setDefault
interfaceEntry config -enable             true
interfaceEntry config -enableDhcp        true
interfaceEntry config -description      "Port 04:01 Interface-1"

# Now add the interface entry to the table
if [interfaceTable addInterface interfaceTypeConnected] {
    logMsg "Error in interfaceEntry addInterface"
    set retCode "FAIL"
}

# Tell the stream to use an interface and the particular interface
stream config -enableSourceInterface     true
stream config -sourceInterfaceDescription "Port 04:01
Interface-1"
if [stream set $ch $ca $po 1] {
    logMsg "Error in interfaceEntry addInterface"
    set retCode "FAIL"
}

# Enable DHCP statistics
stat config -enableDhcpStats            true
stat set $ch $ca $po
stat write $ch $ca $po

# Send the interface table to the chassis
if [interfaceTable write] {
    logMsg "Error in interfaceTable write"
    set retCode "FAIL"
}

# Need to wait until the interface has been defined and
while {1} {
    sleep 200
    stat get allStats $ch $ca $po
    if {1 == [stat cget -dhcpV4EnabledInterfaces]} {
        break
    }
}

```

```

# Need to wait until the DHCP server has assigned an address
interfaceTable requestDiscoveredTable
while {1} {
    sleep 200
    stat get allStats $ch $ca $po
    if {1 == [stat cget -dhcpV4AddressesLearned]} {
        break
    }
}

if [interfaceTable select $ch $ca $po] {
    logMsg "Error selecting $ch $ca $po"
    set retCode "FAIL"
}

# Get the first interface
if [interfaceTable getFirstInterface interfaceTypeConnected] {
    logMsg "Error getFirstInterface $ch $ca $po"
    set retCode "FAIL"
}

# Ask for the discovered DHCP information
if [interfaceTable requestDiscoveredTable] {
    logMsg "Error requestDiscoveredTable $ch $ca $po"
    set retCode "FAIL"
}

# And fetch it - attempts will timeout after 10s
set time_elapsed_ms 0
while {[interfaceTable getDhcpV4DiscoveredInfo "Port 04:01
Interface-
1"]} {
    if {$time_elapsed_ms > 10000} {
        logMsg "Error getDhcpV4DiscoveredInfo $ch $ca $po"
        set retCode "FAIL"
    }
    incr time_elapsed_ms 100
    after 100
}
# Pull out the assigned IP address, mask, gateway and timer
set ipAddress      [dhcpV4DiscoveredInfo cget -ipAddress]
set prefixLength   [dhcpV4DiscoveredInfo cget -prefixLength]
set gatewayIpAddress [dhcpV4DiscoveredInfo cget -gatewayIpAddress]
set renewTimer      [dhcpV4DiscoveredInfo cget -renewTimer]

# Look at the first TLV
if [dhcpV4DiscoveredInfo getFirstTlv] {
    logMsg "Error getFirstTlv $ch $ca $po"
    set retCode "FAIL"
}
set type   [dhcpV4Tlv cget -type]
set value  [dhcpV4Tlv cget -value]

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

discoveredAddress, discoveredList, discoveredNeighbor, interfaceEntry, interfaceIpV4, interfaceIpV6, dhcpV4DiscoveredInfo, dhcpV4Properties, dhcpV4Tlv

NAME - ip

ip - configure the IP parameters for a port on a card on a chassis

SYNOPSIS

ip *sub-command options*

DESCRIPTION

The *ip* command is used to configure the IP-specific information used when building IP-type packets if the **protocol config-name** has been set to *ip*. See RFC 791 for a complete definition of IP header fields. Note that *stream get* must be called before this command's get sub-command.

The TOS byte in the IP header may be interpreted as TOS or DSCP. The options controlling this choice and DSCP settings are *qosMode*, *dscpMode*, *dscpValue*, *classSelector*, *assuredForwardingClass* and *assuredForwardingPrecedence*.

The source and destination addresses may be set from the result of a PPP negotiation through the use of the *enableDestSyncFromPpp* and *enableSourceSyncFromPpp* options. Note that it is necessary to wait until the PPP session has been negotiated before:

- performing a *chassis refresh* command
- performing a *stream get* command
- performing an *ip get* command
- reading the *destAddr* and *sourceAddr* values using *ip cget*

Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in *atmHeader*. The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command is with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

STANDARD OPTIONS

assuredForwarding Class

If *qosMode* is set to *ipv4ConfigDscp* and *dscpMode* is set to *ipV4DscpAssuredForwarding*, then this is the assured forwarding class.

Option	Value	Usage
<i>ipV4DscpClass1</i>	0	(default) Class 1
<i>ipV4DscpClass2</i>	1	Class 2
<i>ipV4DscpClass3</i>	2	Class 3
<i>ipV4DscpClass4</i>	3	Class 4
<i>ipV4DscpClass5</i>	4	Class 5
<i>ipV4DscpClass6</i>	5	Class 6
<i>ipV4DscpClass7</i>	6	Class 7

**assuredForwarding
Precedence**

If *qosMode* is set to *ipv4ConfigDscp* and *dscpMode* is set to *ipV4DscpAssuredForwarding*, then this is the assured forwarding precedence.

Option	Value	Usage
<i>ipV4DscpPrecedenceLowDrop</i>	0	(default) Low drop rate.
<i>ipV4DscpPrecedenceMediumDrop</i>	1	Medium drop rate.
<i>ipV4DscpPrecedenceHighDrop</i>	2	High drop rate.

checksum

Read-only. Value of the checksum in the valid ip stream. Valid only if the **stream set** is performed.

classSelector

If *qosMode* is set to *ipv4ConfigDscp* and *dscpMode* is set to *ipV4DscpClassSelector*, then this holds the class selector value.

Option	Value	Usage
<i>ipV4DscpClass1</i>	0	(default) Class 1
<i>ipV4DscpClass2</i>	1	Class 2
<i>ipV4DscpClass3</i>	2	Class 3
<i>ipV4DscpClass4</i>	3	Class 4
<i>ipV4DscpClass5</i>	4	Class 5
<i>ipV4DscpClass6</i>	5	Class 6
<i>ipV4DscpClass7</i>	6	Class 7

cost

Part of the Type of Service byte of the IP header datagram (bit 6). Options include:

Option	Value	Usage
<i>normalCost</i>	0	(default)
<i>lowCost</i>	1	

delay

Part of the Type of Service byte of the IP header datagram (bit 3). Options include:

Option	Value	Usage
<i>NormalDelay</i>	0	(default)
<i>LowDelay</i>	1	

destClass

Class type associated with the destination IP address of the Ixia port. Options include:

Option	Value	Usage
<i>classA</i>	0	
<i>classB</i>	1	
<i>classC</i>	2	(default)
<i>classD</i>	3	
<i>noClass</i>	4	

destDutIpAddr

IP address of the DUT (device under test) port. This value is stored at the **TclHal** level. (*default = 127.0.0.1*)

destIpAddr

Destination IP address of the Ixia port. (*default = 127.0.0.1*)

destIpAddrMode

Specifies how the destination IP address is incremented or decremented. If **destIpAddrRepeatCount** is set to 1, this variable has no effect. Possible values include:

Option	Value	Usage
<i>ipIdle</i>	0	(<i>default</i>) no change to IP address regardless of destIpAddrRepeatCount
<i>ipIncrHost</i>	1	increment the host portion of the IP address for as many destIpAddrRepeatCount specified
<i>ipDecrHost</i>	2	decrement the host portion of the IP address for as many destIpAddrRepeatCount specified
<i>ipContIncrHost</i>	3	Continuously increment the host portion of the IP address for each packet
<i>ipContDecrHost</i>	4	Continuously decrement the host portion of the IP address for each packet
<i>ipIncrNetwork</i>	5	increment the network portion of the IP address for as many destIpAddrRepeatCount specified
<i>ipDecrNetwork</i>	6	increment the network portion of the IP address for as many destIpAddrRepeatCount specified
<i>ipContIncrNetwork</i>	7	Continuously increment the network portion of the IP address for each packet
<i>ipContDecrNetwork</i>	8	Continuously decrement the network portion of the IP address for each packet.
<i>ipRandom</i>	9	Generate random IP addresses

destIpAddrRepeatCount

Number of destination IP addresses. If set to 1, **destIpAddrMode** has no effect (*default = 1*)

destIpMask

Destination IP subnet mask. (*default = 255.0.0.0*)

destMacAddr

Destination MAC address, generally the MAC address of the DUT port; this field is modified on receipt of ARP frames. This value is stored at the TclHal level. (*default = 00 00 00 00 00 00*)

dscpMode

If *qosMode* is set to *ipv4ConfigDscp*, then this indicates the particular DSCP mode to be used.

Option	Value	Usage
<i>ipV4DscpDefault</i>	0	(<i>default</i>) The default mode of best efforts. No other options apply
<i>ipV4DscpClassSelector</i>	1	Class selector mode. The particular class is set in the <i>classSelector</i> option.
<i>ipV4DscpAssuredForwarding</i>	2	Assured forwarding. The class is set in the <i>assuredForwardingClass</i> option and the assured forwarding precedence is set in the <i>assuredForwardingPrecedence</i> option.
<i>ipV4DscpExpeditedForwarding</i>	3	Expedited forwarding. No other options apply.
<i>ipV4DscpCustom</i>	4	An arbitrary value may be set in the TOS byte, held in <i>dscpValue</i> .

dscpValue

If *qosMode* is set to *ipv4ConfigDscp* and *dscpMode* is set to *ipV4DscpCustom*, then this holds the value of the TOS/DSCP byte.

enableDestSyncFrom Ppp true / false

If *true*, then the *destIpAddr* is set from negotiated PPP session. See the note at the head of this command about interaction with the PPP negotiation process.
(*default = false*)

enableHeaderLength Override true / false

If *false*, then the *headerLength* field is automatically set, based on the Frame Size set in the Frame Control tab. If *true*, then the value may be overridden. (*default = false*)

enableSourceSyncFrom Ppp true / false

If *true*, then the *sourceIpAddr* is set from negotiated PPP session. See the note at the head of this command about interaction with the PPP negotiation process.
(*default = false*)

fragment

If set to *true*, this field indicates this is a fragmented datagram. Used in conjunction with *identifier*, *fragmentOffset* and *lastFragment*. Options include:

Option	Value	Usage
may	0	(<i>default</i>)
dont	1	

fragmentOffset

This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). This differs from the display in IxExplorer, where the fragment offset is displayed in terms of bytes. The first fragment has offset zero. (*default = 0*)

headerLength

Automatically calculated to include the minimum of five 32-bit words plus optional data and padding. (*default = 20*)

identifier

An identifying value assigned by the sender to aid in assembling the fragments of a datagram. (*default = 0*)

ipProtocol

The next level protocol used in the data portion of the internet datagram. Possible values include:

Option	Value	Usage
ipV6HopToHop ipV4ProtocolIpV6HopByHop	0	
icmp ipV4ProtocolIcmp	1	
igmp ipV4ProtocolIgmp	2	
ggp ipV4ProtocolGgp	3	
ipv4ProtocolIpV4	4	
st ipV4ProtocolSt	5	
tcp ipV4ProtocolTcp	6	
ucl ipV4ProtocolUcl	7	

Option	Value	Usage
egp ipV4ProtocolEgp	8	
igp ipV4ProtocolIgp	9	
bbn_rcc_mon ipV4ProtocolBbnRccMon	10	
nvp_ii ipV4ProtocolNvpli	11	
pup ipV4ProtocolPup	12	
argus ipV4ProtocolArgus	13	
emcon ipV4ProtocolEmcon	14	
xnet ipV4ProtocolXnet	15	
chaos ipV4ProtocolChaos	16	
udp ipV4ProtocolUdp	17	(default)
mux ipV4ProtocolMux	18	
dcn_meas ipV4ProtocolDcnMeas	19	
hmp ipV4ProtocolHmp	20	
prm ipV4ProtocolPrm	21	
xns_idp ipV4ProtocolXnsIdp	22	
trunk_1 ipV4ProtocolTrunk1	23	
trunk_2 ipV4ProtocolTrunk2	24	
leaf_1 ipV4ProtocolLeaf1	25	
leaf_2 ipV4ProtocolLeaf2	26	
rdp ipV4ProtocolRdp	27	
irtp ipV4ProtocolIrtp	28	
iso_tp4 ipV4ProtocolIsoTp4	29	
netblt ipV4ProtocolNetblt	30	
mfe_nsp ipV4ProtocolMfeNsp	31	
merit_inp ipV4ProtocolMeritInp	32	

Option	Value	Usage
sep	33	
ipV4ProtocolSep		
ipV4Protocol3pc	34	
ipV4ProtocolDpr	35	
ipV4ProtocolXtp	36	
ipV4ProtocolDdr	37	
ipV4ProtocolDprCmtp	38	
ipV4ProtocolTpPlusPlus	39	
ipV4ProtocolIITransportProtocol	40	
ipV4ProtocolIpv6	41	
ipV4ProtocolSdrp	42	
ipV4ProtocolSipSr	43	
ipV4ProtocolSipFrag	44	
ipV4ProtocolDlrdp	45	
ipV4ProtocolRsvp	46	
ipV4ProtocolGre	47	
ipV4ProtocolMhrp	48	
ipV4ProtocolBna	49	
ipV4ProtocolSippEsp	50	
ipV4ProtocolSippAh	51	
ipV4ProtocolNlsp	52	
ipV4ProtocolSwipe	53	
ipV4ProtocolNarp	54	
ipV4ProtocolMobile	55	
ipV4ProtocolTlsp	56	
ipV4ProtocolSkip	57	
ipV4ProtocolIpv6ICMP	58	
ipV4ProtocolIpv6NoNext	59	
ipV4ProtocolIpv6Opts	60	
ipV4ProtocolHostInternalProtocol	61	
cftp	62	
ipV4ProtocolCftp		
ipV4ProtocolAnyLocalNetwork	63	
sat_expak	64	
ipV4ProtocolSatExpak		
mit_subnet	65	
ipV4ProtocolKriptolan		
rwd	66	
ipV4ProtocolRwd		
ippc	67	
ipV4ProtocolIppc		
ipV4ProtocolAnyDistFileSystem	68	
ipV4ProtocolSatMon	69	
ipV4ProtocolVisa	70	

Option	Value	Usage
ipcv	71	
ipV4ProtocolPvc		
ipV4ProtocolCpx	72	
ipV4ProtocolCphb	73	
ipV4ProtocolWsn	74	
ipV4ProtocolPvp	75	
br_sat_mon	76	
ipV4ProtocolBrSatMon		
ipV4ProtocolSunNd	77	
wb_mon	78	
ipV4ProtocolWbMon		
wb_expak	79	
ipV4ProtocolWbExpak		
ipV4ProtocolSolv	80	
ipV4ProtocolVmtp	81	
ipV4ProtocolSequareVmtp	82	
ipV4ProtocolVines	83	
ipV4ProtocolTtp	84	
ipV4ProtocolNsfnnet	85	
ipV4ProtocolDgp	86	
ipV4ProtocolTcf	87	
ipV4ProtocolEigrp	88	
ipV4ProtocolOspf	89	
ipV4ProtocolSpriteRpc	90	
ipV4ProtocolLarp	91	
ipV4ProtocolMtp	92	
ipV4ProtocolAx25	93	
ipV4ProtocolPip	94	
ipV4ProtocolMicp	95	
ipV4ProtocolSccSp	96	
ipV4ProtocolEtherip	97	
ipV4ProtocolEncap	98	
ipV4ProtocolAnyPrivateEncrScheme	99	
ipV4ProtocolGmtp	100	
ipV4ProtocolIfmp	101	
ipV4ProtocolPnni	102	
ipV4ProtocolPim	103	
ipV4ProtocolAris	104	
ipV4ProtocolScps	105	
ipV4ProtocolQnx	106	
ipV4ProtocolActiveNetwork	107	
ipV4ProtocolIpComp	108	
ipV4ProtocolSnp	109	
ipV4ProtocolCompaqPeer	110	

Option	Value	Usage
ipV4ProtocolPpxInIp	111	
ipV4ProtocolVrrp	112	
ipV4ProtocolAnyZeroHop	113	
ipV4ProtocolL2tp	115	
ipV4ProtocolDdx	116	
ipV4ProtocolLatp	117	
ipV4ProtocolStp	118	
ipV4ProtocolSrp	119	
ipV4ProtocolUti	120	
ipV4ProtocolSmp	121	
ipV4ProtocolSm	122	
ipV4ProtocolPtp	123	
ipV4ProtocolSisis	124	
ipV4ProtocolFire	125	
ipV4ProtocolCrtp	126	
ipV4ProtocolCrudp	127	
ipV4ProtocolSscopmce	128	
ipV4ProtocolPlt	129	
ipV4ProtocolSps	130	
ipV4ProtocolPipe	131	
ipV4ProtocolSctp	132	
ipV4ProtocolFiberChannel	133	
ipV4ProtocolRsvpE2eIgnore	134	
ipV4ProtocolMobilityHeader	135	
ipV4ProtocolUdpLite	136	
ipV4ProtocolMplsInIp	137	

lastFragment

Controls whether there are additional fragments used to assemble this datagram.
Options include:

Option	Value	Usage
last	0	(default)
more	1	

**lengthOverride
*true / false***

If true, enables changing the total length in the ip header. (default = *false*)

options

Variable length option field in the IP header datagram. (default = {})

precedence

Part of the Type of Service byte of the IP header datagram. Establishes precedence of delivery. Possible values are:

Option	Value	Usage
routine	0x0	(default)
priority	0x1	
immediate	0x2	

Option	Value	Usage
flash	0x3	
flashOverride	0x4	
criticEcp	0x5	
internetControl	0x6	
networkControl	0x7	

qosMode

The manner in which the TOS byte is to be interpreted.

Option	Value	Usage
ipV4ConfigTos	0	(default) TOS - type of service.
ipV4ConfigDscp	1	DSCP - DiffSrv.

reliability

Part of the Type of Service byte of the IP header datagram (bit 5). Options include:

Option	Value	Usage
normalReliability	0	(default)
highReliability	1	

reserved

Part of the Type of Service byte of the IP header datagram (bit 7 - 0/1). (default = 0)

sourceClass

Class type associated with the source IP address. Options include:

Option	Value	Usage
classA	0	
classB	1	
classC	2	(default)
classD	3	
noClass	4	

sourceIpAddr

Source IP address. (default = 127.0.0.1)

Note: If the source address equals a DHCP Protocol Interface entry, then the command - *chassis refresh <chassis name>* - must be issued before subsequently issuing the *get* and *cget* commands in the local Tcl client to ensure an accurate reading.

sourceIpAddrMode

Specifies how the source IP address is incremented or decremented. If **sourceIpAddrRepeatCount** is set to 1, this variable has no effect. Possible values include:

Option	Value	Usage
<i>ipIdle</i>	0	(default) no change to IP address regardless of sourceIpAddrRepeatCount
<i>ipIncrHost</i>	1	increment the network portion of the IP address for as many sourceIpAddrRepeatCount specified
<i>ipDecrHost</i>	2	decrement the network portion of the IP address for as many sourceIpAddrRepeatCount specified

Option	Value	Usage
<i>ipContIncrHost</i>	3	Continuously increment the network portion of the IP address for each packet
<i>ipContDecrHost</i>	4	Continuously decrement the network portion of the IP address for each packet
<i>ipIncrNetwork</i>	5	increment the network portion of the IP address for as many sourceIpAddrRepeatCount specified
<i>ipDecrNetwork</i>	6	increment the network portion of the IP address for as many sourceIpAddrRepeatCount specified
<i>ipContIncrNetwork</i>	7	Continuously increment the network portion of the IP address for each packet
<i>ipContDecrNetwork</i>	8	Continuously decrement the network portion of the IP address for each packet.
<i>ipRandom</i>	9	Generate random IP addresses

sourceIpAddrRepeatCount Number of source IP addresses. If set to 1, **sourceAddrMode** has no effect.
(*default = 1*)

sourceIpMask Source IP subnet mask. (*default = 255.0.0.0*)

throughput Part of the Type of Service byte of the IP header datagram (bit 4). Options include:

Option	Value	Usage
<i>normalThruput</i>	0	(<i>default</i>)
<i>highThruput</i>	1	

totalLength Total Length is the length of the datagram, measured in octets, including internet header and data. (*default = 46*)

ttl Time-to-Live, measured in units of seconds. (*default = 64*)

useValidChecksum valid/invalid/override If *portFeatureTcpIPv4ChecksumOverride* = true, then:
Valid: (*default*) The calculated header checksum is automatically calculated.

Invalid: The calculated header checksum is automatically calculated (with error).

Override: The header checksum can be set to a user-defined, 2-octet value.

COMMANDS

The **ip** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ip cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ip** command.

ip config option value

Modify the IP configuration options of the port. If no *option* is specified, returns a list describing all of the available IP options (see STANDARD OPTIONS) for port.

ip decode *capFrame chasID cardID portID*

Decodes a captured frame in the capture buffer and updates TclHal. **ip cget option** command can be used after decoding to get the option data. Specific errors are:

- No connection to a chassis
- Invalid port number
- The captured frame is not a valid IP frame

ip get *chasID cardID portID*

Gets the current IP configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Note that *stream* get must be called before this command's get sub-command. Call this command before calling **ip cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

ip set *chasID cardID portID*

Sets the IP configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **ip config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

ip setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *tcp*

SEE ALSO

stream, protocol, ipx, udp, tcp

NAME - ipAddressTable

ipAddressTable - configure the IP address table parameters for a port on a card on a chassis

SYNOPSIS	ipAddressTable <i>sub-command options</i>
-----------------	---

DESCRIPTION	The <i>ipAddressTable</i> command is used to configure the IP address table-specific information used when building IP address table.
--------------------	---

STANDARD OPTIONS

defaultGateway	Default gateway IP address. (<i>default</i> = 0.0.0.0)
-----------------------	---

COMMANDS	The ipAddressTable command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	---

ipAddressTable addItem

Creates IP and MAC address ranges. Specific errors are:

- The configured parameters are not valid for this port

ipAddressTable cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ipAddressTable** command.

ipAddressTable clear

Clears the IP address table.

ipAddressTable config *option value*

Modify the IP address table configuration options of the port. If no *option* is specified, returns a list describing all of the available ipAddressTable options (see STANDARD OPTIONS) for port.

ipAddressTable delItem

Deletes IP and MAC address ranges.

ipAddressTable get *chasID cardID portID*

Gets the current IP address table configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Call this command before calling **ipAddressTable** cget *option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

ipAddressTable get *chasID cardID portID*

Gets the IP address table configuration of the port with id *portID* on card *cardID*, chassis *chasID*.

ipAddressTable `getFirstItem`

Gets the first IP and MAC address range out of the IP address table. Specific errors are:

- There is no IP address table in the IP server
- There are no more entries in the IP table

ipAddressTable `getItem` *ipAddress*

Gets the address table item corresponding to *ipAddress*. The values may be retrieved by using the *ipAddressTableItem* command. Specific errors are:

- There is no IP address table in the IP server
- There are no more entries in the IP table

ipAddressTable `getNextItem`

Gets the next IP and MAC address range out of the IP address table. Specific errors are:

- There is no IP address table in the IP server
- There are no more entries in the IP table

ipAddressTable `set` *chasID cardID portID*

Sets the IP address table configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the *ipAddressTableItem config option value* command.

ipAddressTable `setDefault`

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *arp*.

SEE ALSO

ip

NAME - ipAddressTableItem

ipAddressTableItem - configure the IP address table parameters for a port on a card on a chassis

SYNOPSIS	ipAddressTableItem <i>sub-command options</i>
-----------------	---

DESCRIPTION	The <i>ipAddressTableItem</i> command is used to configure the IP address table-specific information used when building IP address table.
--------------------	---

STANDARD OPTIONS

atmEncapsulation For ATM type load modules, this is the type of ATM encapsulation that is used on the interface.

Option	Value	Usage
atmEncapsulationVccMuxIPV4Routed	101	
atmEncapsulationVccMuxBridgedEthernetFCS	102	
atmEncapsulationVccMuxBridgedEthernetNoFCS	103	
atmEncapsulationVccMuxIPV6Routed	104	
atmEncapsulationVccMuxMPLSRouted	105	
atmEncapsulationLLCRoutedCLIP	106	
atmEncapsulationLLCBridgedEthernetFCS	107	(<i>default</i>)
atmEncapsulationLLCBridgedEthernetNoFCS	108	
atmEncapsulationLLCPPPoA	109	
atmEncapsulationVccMuxPPPoA	110	

atmVci

The ATM VCI number, if this is an ATM port. (*default = 0*)

atmVpi

The ATM VPI number, if this is an ATM port. (*default = 0*)

enableExpandInterfaceTable *true / false*

If *true*, then the range of IP addresses from *fromIpAddress* to *toIpAddress* are expanded to individual IP addresses on the port. This only operates on ports with individual CPUs and is for internal use only. (*default = false*)

enableUseNetwork
true / false

If set, the *netMask* field is used to set the network mask; otherwise, the network mask is 0.0.0.0. (*default = false*)

enableVlan
true / false

Enables VLAN encapsulation of routing protocols. The VLAN ID is in the *vlanId* option. (*default = false*)

fromIpAddress

The first IP address for the IP address range. (*default = 0.0.0.0*)

fromMacAddress

The first MAC address for the MAC address range. (*default = {00 00 00 00 00 00}*)

gatewayIpAddress

Default gateway IP address. (*default = 0.0.0.0*)

mappingOption

Specifies the mapping option.

Option	Value	Usage
oneIpToOneMAC	0	(default)
manyIpToOneMAC	1	

netMask

If *enableUseNetwork* is set, this value is used to set the network mask. (*default* = 24).

numAddresses

Number of consecutive addresses. (*default* = 1)

**overrideDefault
Gateway true/false**

Enable default gateway IP address. (*default* = false)

toIpAddress

Read-Only. Last IP address in the IP address range. (*default* = 0.0.0.0)

toMacAddress

Read-Only. Last MAC address in the MAC address range. (*default* = 00 00 00 00 00 00)

vlanId

If *enableVlan* is true, the routing protocols are VLAN encapsulated with this ID. Although a value of '0' is allowed, VLAN IDs normally start at 1. (*default* = 0)

COMMANDS

The **ipAddressTableItem** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ipAddressTableItem cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ipAddressTableItem** command.

ipAddressTableItem config option value

Modify the IP address table configuration options of the port. If no *option* is specified, returns a list describing all of the available ipAddressTableItem options (see STANDARD OPTIONS) for port.

ipAddressTableItem get

Gets the current IP address table item configuration. Call this command before calling **ipAddressTableItem cget option value** to get the value of the configuration option.

ipAddressTableItem set

Sets the IP address table item configuration, by reading the configuration option values set by the **ipAddressTableItem config option value** command.

ipAddressTableItem setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under [arp](#).

SEE ALSO

[ipAddressTable](#)

NAME - ipV6

ipV6 - configure the ipV6 options for a stream

SYNOPSIS	ipV6 <i>sub-command options</i>
DESCRIPTION	<p>The <i>ipV6</i> command is used to configure the IPv6 options associated with a stream a <i>stream set</i> and <i>stream write</i> must follow an <i>ipV6 set</i> command. The <i>ipV6</i> object also includes list of extension headers. Extension headers are built-in with type specific objects:</p> <ul style="list-style-type: none"> • ipV6Authentication on page A-352 • ipV6Destination on page A-354 • ipV6Fragment on page A-356 • ipV6Routing on page A-373 • ipV6HopByHop on page A-358 <p>An extension header is added to a <i>ipV6</i> object by configuring the extension header with the appropriate command from the list above and then adding it to the group with <i>ipV6 addExtensionHeader</i> <i>type</i>, where <i>type</i> indicates which of the extensions to use. An extension may be retrieved from an <i>ipV6</i> object through the use of <i>getFirstExtensionHeader</i> / <i>getNextExtensionHeader</i>. These commands return the name of the command that was used to configure the header extension. The type of the extension header can be determined from the <i>nextHeader</i> value from the <i>ipV6</i> command (for the first extension header) or from the previous extension header otherwise. This is typically used in the following sequence of commands:</p> <pre>set eHeader [ipV6 getFirstExtensionHeader] set nextType [\$eHeader cget -nextHeader]</pre> <p>In addition, if this is to be the header to a TCP, UDP or ICMP packet, then a separate call to <i>ipV6 addExtensionHeader</i> must be made with <i>tcp</i>, <i>udp</i> or <i>icmpV6</i> must be made. For example:</p> <pre>ipV6 addExtensionHeader tcp</pre> <p>Although it is the default, <i>ipV6 addExtensionHeader ipV6NoNextHeader</i> may be used to indicate that there is no header following this one.</p> <p>Note that <i>stream</i> <i>get</i> must be called before this command's <i>get</i> sub-command. The source and destination addresses may be set from the result of a PPP negotiation through the use of the <i>enableDestSyncFromPpp</i> and <i>enableSourceSyncFromPpp</i> options. Note that it is necessary to wait until the PPP session has been negotiated before:</p> <ul style="list-style-type: none"> • performing a <i>chassis refresh</i> command • performing a <i>stream get</i> command • performing an <i>ipV6 get</i> command • reading the <i>destAddr</i> and <i>sourceAddr</i> values using <i>ipV6 cget</i>
<hr/>	

STANDARD OPTIONS

destAddr	The destination address, expressed as any valid IPv6 format address. (<i>default = {0:0:0:0:0:0:0:0}</i>)
-----------------	---

destAddrMode

The manner in which the destination address is modified per packet. For all but the default case, one of the UDFs is reserved for this use.

Option	Value	Usage	Valid with Address Prefix:
<i>ipV6Idle</i>	0	(default) No change to address.	All
<i>ipV6IncrHost</i>	1	Increment the host part of the address (as indicated by <i>sourceMask</i>) by <i>sourceStepSize</i> for <i>sourceAddrRepeatCount</i> before restarting at the <i>sourceAddr</i> value.	Reserved, NSAP Allocation, IPX Allocation, User Defined
<i>ipV6DecrHost</i>	2	Decrement the host part of the address (as indicated by <i>sourceMask</i>) by <i>sourceStepSize</i> for <i>sourceAddrRepeatCount</i> before restarting at the <i>sourceAddr</i> value.	Reserved, NSAP Allocation, IPX Allocation, User Defined
<i>ipV6IncrNetwork</i>	3	Increment the network part of the address (as indicated by <i>sourceMask</i>) by <i>sourceStepSize</i> for <i>sourceAddrRepeatCount</i> before restarting at the <i>sourceAddr</i> value.	Reserved, NSAP Allocation, IPX Allocation, User Defined
<i>ipV6DecrNetwork</i>	4	Increment the network part of the address (as indicated by <i>sourceMask</i>) by <i>sourceStepSize</i> for <i>sourceAddrRepeatCount</i> before restarting at the <i>sourceAddr</i> value.	Reserved, NSAP Allocation, IPX Allocation, User Defined
<i>ipV6IncrInterfaceId</i>	5	For use when the address is Link Local Unicast, Site Local Unicast or Global Unicast. Increments the interface ID part of the address.	Global Unicast, Link Local Unicast, Site Local Unicast
<i>ipV6DecrInterfaceId</i>	6	For use when the address is Link Local Unicast, Site Local Unicast or Global Unicast. Decrements the interface ID part of the address.	Global Unicast, Link Local Unicast, Site Local Unicast
<i>ipV6IncrGlobalUnicastTopLevelAggrId</i>	7	For use when the address is Global Unicast. Increments the top level aggregation ID part of the address.	Global Unicast
<i>ipV6DecrGlobalUnicastTopLevelAggrId</i>	8	For use when the address is Global Unicast. Decrements the top level aggregation ID part of the address.	Global Unicast
<i>ipV6IncrGlobalUnicastNextLevelAggrId</i>	9	For use when the address is Global Unicast. Increments the next level aggregation ID part of the address.	Global Unicast
<i>ipV6DecrGlobalUnicastNextLevelAggrId</i>	10	For use when the address is Global Unicast. Decrements the next level aggregation ID part of the address.	Global Unicast
<i>ipV6IncrGlobalUnicastSiteLevelAggrId</i>	11	For use when the address is Global Unicast. Increments the site level aggregation ID part of the address.	Global Unicast
<i>ipV6DecrGlobalUnicastSiteLevelAggrId</i>	12	For use when the address is Global Unicast. Decrements the site level aggregation ID part of the address.	Global Unicast
<i>ipV6IncrSiteLocalUnicastSubnetId</i>	13	For use when the address is Site Local Unicast. Increments the Subnet ID part of the address.	Site Local Unicast
<i>ipV6DecrSiteLocalUnicastSubnetId</i>	14	For use when the address is Site Local Unicast. Decrements the Subnet ID part of the address.	Site Local Unicast

Option	Value	Usage	Valid with Address Prefix:
<i>ipV6IncrMulticastGroupId</i>	15	For use when the address is Multicast. Increments the multicast group part of the address.	Multicast
<i>ipV6DecrMulticastGroupId</i>	16	For use when the address is Multicast. Decrements the multicast group part of the address.	Multicast
<i>ipV6IncrementGlobalUnicastGlobalRoutingPrefixId</i>	17	Increments the corresponding field of the new Global Unicast 3587 address mode	Global Unicast 3587
<i>ipV6DecrementGlobalUnicastGlobalRoutingPrefixId</i>	18	Increments the corresponding field of the new Global Unicast 3587 address mode	Global Unicast 3587
<i>ipV6IncrementSubnetId</i>	19	Increments the corresponding field of the new Global Unicast 3587 address mode	Global Unicast 3587
<i>ipV6DecrementSubnetId</i>	20	Increments the corresponding field of the new Global Unicast 3587 address mode	Global Unicast 3587

destAddrRepeatCount

The number of times to repeat the function indicated in *destAddrMode* (except *ipV6Idle*) before restarting the address at *destAddr*. (*default = 10*)

destMask

The number of bits in the network mask part of the address, counting from the high-order bits. For use with *destAddrMode* set to all but *ipV6Idle* mode.

This command's valid range is dependent on what options is selected in *destAddrMode* above:

Option	Range
<i>decrMulticastGroupId</i>	fixed at 96
<i>incrMulticastGroupId</i>	fixed at 96
<i>decrGlobalUnicastTopLevelAggregationId</i>	fixed at 4
<i>incrGlobalUnicastTopLevelAggregationId</i>	fixed at 4
<i>decrGlobalUnicastNextLevelAggregationId</i>	fixed at 24
<i>incrGlobalUnicastNextLevelAggregationId</i>	fixed at 24
<i>decrGlobalUnicastSiteLevelAggregationId</i>	fixed at 48
<i>incrGlobalUnicastSiteLevelAggregationId</i>	fixed at 48
<i>decrSiteLocalUnicastSubnetId</i>	fixed at 48
<i>incrSiteLocalUnicastSubnetId</i>	fixed at 48
<i>incrHost</i>	96 to 128
<i>decrHost</i>	96 to 128
<i>decrNetwork</i>	96 to 128
<i>incrNetwork</i>	96 to 128
<i>decrInterfaceId</i>	96 to 128
<i>incrInterfaceId</i>	96 to 128
<i>ipIncrmentGlobalUnicastGlobalRoutingPrefixId</i>	16 to 48
<i>ipDecrementGlobalUnicastGlobalRoutingPrefixId</i>	16 to 48
<i>ipIncrmentSubnetId</i>	Range- fixed at 48
<i>ipDecrementSubnetId</i>	Range- fixed at 48

destStepSize

The amount to increment the address by between iterations. For use with *destAddrMode* set to all but *ipV6Idle* mode. (*default = 1*)

enableDestSyncFrom
Ppp true / false

If *true*, then the *destAddr* is set from negotiated PPP session. See the note at the head of this command about interaction with the PPP negotiation process.
(default = false)

enableSourceSyncFrom
Ppp true / false

If *true*, then the *sourceAddr* is set from negotiated PPP session. See the note at the head of this command about interaction with the PPP negotiation process.
(default = false)

flowLabel

The flow label for the IPv6 address. *(default = 0)*

hopLimit

The hop limit for the IPv6 address. *(default = 255)*

nextHeader

The type of the next packet header.

Option	Value	Usage
<i>ipV6HopByHopOptions</i>	0	Next header is hop-by-hop options.
<i>ipV6Routing</i>	43	Next header has routing options.
<i>ipV6Fragment</i>	44	Payload is a fragment.
<i>ipV6EncapsulatingSecurityPayload</i>	50	Next header is an IPSEC ESP.
<i>ipV6Authentiication</i>	51	Next header is an IPSEC AH.
<i>ipV6NoNextHeader</i>	59	There is no next header.
<i>ipV6DestinationOptions</i>	60	Next header has destination options.
<i>tcp</i>	6	Next header is TCP.
<i>udp</i>	17	Next header is UDP.
<i>icmpV6</i>	58	Next header is ICMP V6.

payloadLength

Read-only. The calculated payload length.

sourceAddr

The source address, expressed as any valid IPv6 format address. *(default = {0:0:0:0:0:0:0:0})*

sourceAddrMode

The manner in which the source address is modified per packet. For all but the default case, one of the UDFs is reserved for this use.

Option	Value	Usage	Valid with Address Prefix:
<i>ipV6Idle</i>	0	<i>(default)</i> No change to address.	All
<i>ipV6IncrHost</i>	1	Increment the host part of the address (as indicated by <i>sourceMask</i>) by <i>sourceStepSize</i> for <i>sourceAddrRepeatCount</i> before restarting at the <i>sourceAddr</i> value.	Reserved, NSAP Allocation, IPX Allocation, User Defined
<i>ipV6DecrHost</i>	2	Decrement the host part of the address (as indicated by <i>sourceMask</i>) by <i>sourceStepSize</i> for <i>sourceAddrRepeatCount</i> before restarting at the <i>sourceAddr</i> value.	Reserved, NSAP Allocation, IPX Allocation, User Defined
<i>ipV6IncrNetwork</i>	3	Increment the network part of the address (as indicated by <i>sourceMask</i>) by <i>sourceStepSize</i> for <i>sourceAddrRepeatCount</i> before restarting at the <i>sourceAddr</i> value.	Reserved, NSAP Allocation, IPX Allocation, User Defined
<i>ipV6DecrNetwork</i>	4	Increment the network part of the address (as indicated by <i>sourceMask</i>) by <i>sourceStepSize</i> for <i>sourceAddrRepeatCount</i> before restarting at the <i>sourceAddr</i> value.	Reserved, NSAP Allocation, IPX Allocation, User Defined

Option	Value	Usage	Valid with Address Prefix:
<i>ipV6IncrInterfaceId</i>	5	For use when the address is Link Local Unicast, Site Local Unicast or Global Unicast. Increments the interface ID part of the address.	Global Unicast, Link Local Unicast, Site Local Unicast
<i>ipV6DecrInterfaceId</i>	6	For use when the address is Link Local Unicast, Site Local Unicast or Global Unicast. Decrements the interface ID part of the address.	Global Unicast, Link Local Unicast, Site Local Unicast
<i>ipV6IncrGlobalUnicastTopLevelAggrId</i>	7	For use when the address is Global Unicast. Increments the top level aggregation ID part of the address.	Global Unicast
<i>ipV6DecrGlobalUnicastTopLevelAggrId</i>	8	For use when the address is Global Unicast. Decrements the top level aggregation ID part of the address.	Global Unicast
<i>ipV6IncrGlobalUnicastNextLevelAggrId</i>	9	For use when the address is Global Unicast. Increments the next level aggregation ID part of the address.	Global Unicast
<i>ipV6DecrGlobalUnicastNextLevelAggrId</i>	10	For use when the address is Global Unicast. Decrements the next level aggregation ID part of the address.	Global Unicast
<i>ipV6IncrGlobalUnicastSiteLevelAggrId</i>	11	For use when the address is Global Unicast. Increments the site level aggregation ID part of the address.	Global Unicast
<i>ipV6DecrGlobalUnicastSiteLevelAggrId</i>	12	For use when the address is Global Unicast. Decrements the site level aggregation ID part of the address.	Global Unicast
<i>ipV6IncrSiteLocalUnicastSubnetId</i>	13	For use when the address is Site Local Unicast. Increments the Subnet ID part of the address.	Site Local Unicast
<i>ipV6DecrSiteLocalUnicastSubnetId</i>	14	For use when the address is Site Local Unicast. Decrements the Subnet ID part of the address.	Site Local Unicast
<i>ipV6IncrMulticastGroupId</i>	15	For use when the address is Multicast. Increments the multicast group part of the address.	Multicast
<i>ipV6DecrMulticastGroupId</i>	16	For use when the address is Multicast. Decrements the multicast group part of the address.	Multicast
<i>ipV6IncrementGlobalUnicastGlobalRoutingPrefixId</i>	17	Increments the corresponding field of the new Global Unicast 3587 address mode	Global Unicast 3587
<i>ipV6DecrementGlobalUnicastGlobalRoutingPrefixId</i>	18	Decrements the corresponding field of the new Global Unicast 3587 address mode	Global Unicast 3587
<i>ipV6IncrementSubnetId</i>	19	Increments the corresponding field of the new Global Unicast 3587 address mode	Global Unicast 3587
<i>ipV6DecrementSubnetId</i>	20	Decrements the corresponding field of the new Global Unicast 3587 address mode	Global Unicast 3587

sourceAddrRepeatCount The number of times to repeat the function indicated in *sourceAddrMode* (except *ipV6Idle*) before restarting the address at *sourceAddr*. (*default = 10*)

sourceMask The number of bits in the network mask part of the address, counting from the high-order bits. For use with *sourceAddrMode* set to all but *ipV6Idle* mode.

This command's valid range is dependent on what options is selected in destAddrMode above:

Option	Range
<i>decrMulticastGroupId</i>	fixed at 96
<i>incrMulticastGroupId</i>	fixed at 96
<i>decrGlobalUnicastTopLevelAggregationId</i>	fixed at 4
<i>incrGlobalUnicastTopLevelAggregationId</i>	fixed at 4
<i>decrGlobalUnicastNextLevelAggregationId</i>	fixed at 24
<i>incrGlobalUnicastNextLevelAggregationId</i>	fixed at 24
<i>decrGlobalUnicastSiteLevelAggregationId</i>	fixed at 48
<i>incrGlobalUnicastSiteLevelAggregationId</i>	fixed at 48
<i>decrSiteLocalUnicastSubnetId</i>	fixed at 48
<i>incrSiteLocalUnicastSubnetId</i>	fixed at 48
<i>incrHost</i>	96 to 128
<i>decrHost</i>	96 to 128
<i>decrNetwork</i>	96 to 128
<i>incrNetwork</i>	96 to 128
<i>decrInterfaceId</i>	96 to 128
<i>incrInterfaceId</i>	96 to 128
<i>ipIncrementGlobalUnicastGlobalRoutingPrefixId</i>	16 to 48
<i>ipDecrementGlobalUnicastGlobalRoutingPrefixId</i>	16 to 48
<i>ipIncrementSubnetId</i>	Range- fixed at 48
<i>ipDecrementSubnetId</i>	Range- fixed at 48

sourceStepSize

The amount to increment the address by between iterations. For use with *sourceAddrMode* set to all but *ipV6Idle* mode. (*default = 1*)

trafficClass

The traffic class for the ipV6 address. (*default = 3*)

COMMANDS

The **ipV6** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ipV6 addExtensionHeader *type*

Adds an extension header of the type indicated in the *type* argument. The data for the extension is read from the object that corresponds to the type.

Option	Value	Usage
<i>ipV6HopByHopOptions</i>	0	Data is read from <i>ipV6HopByHop</i> .
<i>ipV6Routing</i>	43	Data is read from <i>ipV6Routing</i> .
<i>ipV6Fragment</i>	44	Data is read from <i>ipV6Fragment</i> .
<i>ipV6EncapsulatingSecurityPayload</i>	50	Not supported in the current release.
<i>ipV6Authentication</i>	51	Data is read from <i>ipV6Authentication</i> .
<i>ipV6DestinationOptions</i>	60	Data is read from <i>ipV6Destination</i> .
<i>ipV6NoNextHeader</i>	59	(<i>default</i>) There is no next header.
<i>tcp</i>	6	Next header is TCP.

Option	Value	Usage
<i>udp</i>	17	Next header is UDP.
<i>icmpV6</i>	58	Next header is ICMP V6.

ipV6 cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ipV6** command.

ipV6 clearAllExtensionHeaders

Removes all of the extension headers from the list.

ipV6 config option value

Modify the *ipV6* options. If no *option* is specified, returns a list describing all of the available *ipV6* options (see STANDARD OPTIONS) for port.

ipV6 decode capFrame [chasID cardID portID]

Decodes a captured frame in the capture buffer and updates TclHal. **ipV6 cget option** command can be used after decoding to get the option data.

ipV6 delExtensionHeader

Deletes the currently referenced extension header accessed through the use of *getFirstExtensionHeader* / *getNextExtensionHeader*. Specific errors include:

- No current header has been accessed.

ipV6 get chasID cardID portID

Gets the current *ipV6* options for the indicated port. Note that *stream* get must be called before this command's get sub-command. Call this command before calling **ipV6 cget option value** to get the value of the configuration option.

ipV6 getFirstExtensionHeader

Access the first extension header in the list. The results of the command is the name of the command used to make the extension header. This command may be symbolically used to view/modify the extension header contents. The type of the extension header is determined from the *nextHeader* value from the *ipV6* command (for the first extension header) or from the previous extension header otherwise. Note that the use of the *addExtensionHeader* sub-command for the *tcp*, *udp*, *icmpV6* and *ipV6NoNextHeader* options does not result in a list element. In the current release, the *IxExplorer* tool allows extension headers of the type *ipv6HopBHopOptions* to be placed in the list. An attempt to retrieve such a header results in no element retrieval and the remainder of the list is inaccessible. Specific errors are:

- There are no extension headers in the list

ipV6 getNextExtensionHeader

Access the next header extension in the list. See the notes and errors in the *getFirstExtensionHeader* sub-command.

ipV6 set *chasID cardID portID*

Sets the ipV6 options by reading the configuration option values set by the **ipV6 config *option value*** command. This command should be followed by a *stream set* and *stream write* commands.

ipV6 setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package req IxTclHal

chassis add thebrain

set chasId      [chassis cget -id]
set cardId      2
set portId      3
set streamId    1

stream setDefault
stream config -framesize 200

# Configure protocol
protocol          setDefault
protocol          config      -name
ipV6
protocol          config      -ethernetType
ethernetII

# Configure ipV6
ipV6              setDefault
ipV6          config      -trafficClass
3
ipV6          config      -sourceAddr
{1:2:3:0:0:0:0:0}
ipV6          config      -sourceMask
64
ipV6          config      -sourceAddrMode
ipV6Idle
ipV6          config      -sourceStepSize
1
ipV6          config      -sourceAddrRepeatCount
10
ipV6          config      -destAddr
{4:5:6:0:0:0:0:0}

# Clear all the extension headers
ipV6          clearAllExtensionHeaders

# Configure and add ipV6Routing extension header
ipV6Routing  setDefault
ipV6Routing  config      -reserved {88 88 88 88}
ipV6Routing  config      -nodeList
{7777:7777:7777:7777:7777:7777:7777,
8888:8888:8888:8888:8888:8888:8888}
if {[ipV6 addExtensionHeader ipV6Routing]} {
    ixPuts "Error adding ipV6Routing "
}

# Configure and add ipV6DestinationOptions extension header
ipV6Destination  setDefault
```

```

if {[ipV6 addExtensionHeader ipV6DestinationOptions]} {
    ixPuts "Error adding ipV6DestinationOptions"
}

# Configure and add ipV6Fragment extension header
ipV6Fragment           setDefault
ipV6Fragment           config      -enableFlag
false
ipV6Fragment           config      -fragmentOffset
345
ipV6Fragment           config      -identification
345
ipV6Fragment           config      -res
1
ipV6Fragment           config      -reserved
170
if {[ipV6 addExtensionHeader ipV6Fragment]} {
    ixPuts "Error adding ipV6Fragment"
}

# Configure and add ipV6Authentication extension header
ipV6Authentication setDefault
ipV6Authentication config -payloadLength      8
ipV6Authentication config -securityParamIndex 1212
ipV6Authentication config -sequenceNumberField 3434
ipV6Authentication config -authentication {44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44}
if {[ipV6 addExtensionHeader ipV6Authentication]} {
    ixPuts "Error adding ipV6Authentication"
}

ipV6HopByHop           clearAllOptions
ipV6OptionPADN         setDefault
ipV6OptionPADN         config      -length
4
ipV6OptionPADN         config      -value
"11 11 11 11"
ipV6HopByHop           addOption    ipV6OptionPADN

ipV6OptionJumbo         setDefault
ipV6OptionJumbo         config      -length
4
ipV6OptionJumbo         config      -payload
5
ipV6HopByHop           addOption    ipV6OptionJumbo

ipV6OptionPADN         setDefault
ipV6OptionPADN         config      -length
6
ipV6OptionPADN         config      -value
"22 22 22 22 22 22"
ipV6HopByHop           addOption    ipV6OptionPADN

ipV6OptionPADN         setDefault
ipV6OptionPADN         config      -length
2
ipV6OptionPADN         config      -value
"00 00"
ipV6HopByHop           addOption    ipV6OptionPADN

ipV6HopByHop           addOption    ipV6OptionPAD1

ipV6OptionPADN         setDefault

```

```

ipV6OptionPADN      config      -length
5
ipV6OptionPADN      config      -value
"33 33 33 33 33"
ipV6HopByHop        addOption   ipV6OptionPADN

ipV6OptionRouterAlert setDefault
ipV6OptionRouterAlert config      -length
2
ipV6OptionRouterAlert config      -routerAlert
ipV6RouterAlertRSVP
ipV6HopByHop        addOption   ipV6OptionRouterAlert

ipV6OptionPADN      setDefault
ipV6OptionPADN      config      -length
5
ipV6OptionPADN      config      -value
"44 44 44 44 44"
ipV6HopByHop        addOption   ipV6OptionPADN

ipV6OptionPADN      setDefault
ipV6OptionPADN      config      -length
2
ipV6OptionPADN      config      -value
"88 88"
ipV6HopByHop        addOption   ipV6OptionPADN

ipV6OptionPADN      setDefault
ipV6OptionPADN      config      -length
1
ipV6OptionPADN      config      -value
00
ipV6HopByHop        addOption   ipV6OptionPADN

ipV6OptionBindingUpdate setDefault
ipV6OptionBindingUpdate config      -length
10
ipV6OptionBindingUpdate config      -acknowledge
1
ipV6OptionBindingUpdate config      -home
1
ipV6OptionBindingUpdate config      -router
1
ipV6OptionBindingUpdate config      -duplicate
1
ipV6OptionBindingUpdate config      -MAP
1
ipV6OptionBindingUpdate config      -bicasting
1
ipV6OptionBindingUpdate config      -prefixLength
5
ipV6OptionBindingUpdate config      -sequenceNumber
5
ipV6OptionBindingUpdate config      -lifeTime
5
ipV6HopByHop        addOption
ipV6OptionBindingUpdate

ipV6OptionPADN      setDefault
ipV6OptionPADN      config      -length
4
ipV6OptionPADN      config      -value
"30 45 45 45"

```

```

ipV6HopByHop           addOption          ipV6OptionPADN
ipV6OptionBindingAck   setDefault        config -length
ipV6OptionBindingAck   config           -status
13
ipV6OptionBindingAck   config           -sequenceNumber
4
ipV6OptionBindingAck   config           -lifeTime
40
ipV6OptionBindingAck   config           -refresh
4
ipV6OptionBindingAck   config           rprVendorSpecific
4
ipV6HopByHop           addOption          ipV6OptionPADN
ipV6OptionPADN         setDefault        config -length
ipV6OptionPADN         config           -value
4
ipV6OptionPADN         "44 44 44 44"
ipV6HopByHop           addOption          ipV6OptionPADN
ipV6OptionPADN         setDefault        config -length
ipV6OptionPADN         config           -value
3
ipV6OptionPADN         "00 00 00"
ipV6HopByHop           addOption          ipV6OptionPADN
ipV6OptionHomeAddress  setDefault        config -length
ipV6OptionHomeAddress  config           -address
14
ipV6OptionHomeAddress  config           "1111:1111:1111:1111:1111:1111:1111:3"
ipV6HopByHop           addOption          ipV6OptionHomeAddress
ipV6OptionPADN         setDefault        config -length
ipV6OptionPADN         config           -value
4
ipV6OptionPADN         "22 22 22 16"
ipV6HopByHop           addOption          ipV6OptionPADN
ipV6OptionBindingRequest setDefault      config -length
ipV6OptionBindingRequest config           -length
9
ipV6HopByHop           addOption          ipV6OptionBindingRequest
ipV6OptionPADN         setDefault        config -length
ipV6OptionPADN         config           -value
4
ipV6OptionPADN         "00 00 00 00"
ipV6HopByHop           addOption          ipV6OptionPADN
ipV6OptionMIPv6UniqueIdSub setDefault      config -length
ipV6OptionMIPv6UniqueIdSub config           -subUniqueId
24
ipV6OptionMIPv6UniqueIdSub config           -subUniqueId
89
ipV6HopByHop           addOption          ipV6OptionMIPv6UniqueIdSub
ipV6OptionMIPv6UniqueIdSub

```

```

ipV6OptionPADN      setDefault
ipV6OptionPADN      config      -length
    2
ipV6OptionPADN      config      -value
    "10 13"
ipV6HopByHop        addOption   ipV6OptionPADN

ipV6OptionMIPv6AlternativeCoaSub setDefault
ipV6OptionMIPv6AlternativeCoaSub config      -length
20
ipV6OptionMIPv6AlternativeCoaSub config      -address
"1414:1414:1414:1414:1414:1414:0:5"
ipV6HopByHop        addOption   ipV6OptionMIPv6AlternativeCoaSub

ipV6OptionPADN      setDefault
ipV6OptionPADN      config      -length
    4
ipV6OptionPADN      config      -value
    "00 00 00 00"
ipV6HopByHop        addOption   ipV6OptionPADN

if {[ipV6 addExtensionHeader ipV6HopByHopOptions]} {
    ixPuts "Error adding ipV6Authentication"
}

# Add tcp
if {[addExtensionHeader ipV4ProtocolTcp]} {
    ixPuts "Error adding tcp"
}

if {[ipV6 set $chassisId $cardId $portId]} {
    ixPuts "Error setting ipV6 on port $chassisId.$cardId.$portId"
}

# Configure tcp
tcp setDefault
tcp config -offset      5
tcp config -sourcePort   16
tcp config -destPort     26
tcp config -useValidChecksum true

if {[tcp set $chassisId $cardId $portId]} {
    ixPuts "Error setting tcp on port $chassisId.$cardId.$portId"
}

# Set and write the stream
if {[stream set $chassisId $cardId $portId $streamId]} {
    ixPuts "Error setting stream $streamId on port
$chassisId.$cardId.$portId"
}

if {[stream write $chassisId $cardId $portId $streamId]} {
    ixPuts "Error writing stream $streamId on port
$chassisId.$cardId.$portId"
}

```

SEE ALSO

stream, ipV6Authentication, ipV6Destination, ipV6Fragment, ipV6Routing,

NAME - ipV6Address

ipV6Address - decode or encode an IPv6 address

SYNOPSIS

ipV6Address *sub-command options*

DESCRIPTION

The *ipV6Address* command is used to create an IPv6 address from component parameters or decode an existing address into its parameters. The *prefixType* of the address dictates which other options are read/written. The following table indicates the options used for each *prefixType* value.

Option	<i>prefixType</i> Values							
	Reserved (0)	NSAP Allocaton (1)	IPX Allocation (2)	Global Unicast (3)	Link Local Unicast (4)	Site Local Unicast (5)	Multicast (6)	User Defined (7)
reservedAddressType	X							
reservedIpv4Address	X							
allocation		X	X					
topLevelAggregationId				X				
reserved				X				
nextLevelAggregationId				X				
siteLevelAggregationId				X				
interfaceId				X	X	X		
subnetId						X		
nonPermanentlyAssigned							X	
scope							X	
groupId							X	
userDefinedAddress								X

STANDARD OPTIONS

allocation	(<i>default = 0</i>)
groupId	(<i>default = 0</i>)
interfaceId	(<i>default = 0</i>)
nextLevelAggregation Id	(<i>default = 0</i>)
nonPermanently Assigned	(<i>default = 0</i>)

STANDARD OPTIONS**for Ipv6 global
Unicast 3587****subnetId** *(default = 0)***interfaceId** *(default = 0)***globalRoutingPrefix** *(default = 0)***prefixType** One of

Option	Value	Usage
<i>ipV6Reserved</i>	0	<i>(default) Reserved.</i>
<i>ipV6NSAPAllocation</i>	1	NSAP Allocation.
<i>ipV6IPXAllocation</i>	2	IPX Allocation.
<i>ipV6GlobalUnicast</i>	3	Global Unicast.
<i>ipV6LinkLocalUnicast</i>	4	Link Local Unicast.
<i>ipV6SiteLocalUnicast</i>	5	Site Local Unicast.
<i>ipV6Multicast</i>	6	Multicast.
<i>ipV6UserDefined</i>	7	User Defined.
<i>ipV6GlobalUnicast3587</i>	8	New global unicast RFC

reserved *(default = 0)***reservedAddressType** One of

Option	Value	Usage
<i>ipV6ReservedCompatible</i>	0	<i>(default) IPv4 compatible address.</i>
<i>ipV6ReservedCompatible</i>	1	IPv4 mapped IPv6 address.

reservedIpV4Address *(default = 0)***scope** One of

Option	Value	Usage
<i>ipV6MulticastScopeReserved1</i>	0	<i>(default)</i>
<i>ipV6MulticastScopeNodeLocalScope</i>	1	Node local scope
<i>ipV6MulticastScopeNodeLinkipV6LocalScope</i>	2	Link local scope
<i>ipV6MulticastScopeUnassigned</i>	3	
<i>ipV6MulticastScopeSiteLocalScope</i>	5	Site local scope
<i>ipV6MulticastScopeOrganizationLocalScope</i>	8	Organization local scope
<i>ipV6MulticastScopeGlobalScope</i>	14	Global scope
<i>ipV6MulticastScopeReserved2</i>	15	

siteLevelAggregationId *(default = 0)*

subnetId	<i>(default = 0)</i>
topLevelAggregationId	<i>(default = 0)</i>
userDefinedAddress	<i>(default = 0)</i>

COMMANDS

The **ipV6Address** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ipV6Address cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6Address* command.

ipV6Address config *option value*

Modify the *ipV6Address* configuration options. If no *option* is specified, returns a list describing all of the available *ipV6Address* options (see STANDARD OPTIONS).

ipV6Address decode *ipV6Address*

Decodes the colon encoded IPv6 address present in *ipv6Address* into the **STANDARD OPTIONS**.

ipV6Address encode

Encodes the IPv6 address present in the *STANDARD OPTIONS* and returns that value as a “`:`” encoded character string.

EXAMPLES

```

package require IxTclHal

ipV6Address setDefault

ipV6Address config -prefixType ipV6GlobalUnicast
ipV6Address config -topLevelAggregationId 10
ipV6Address config -nextLevelAggregationId 42
ipV6Address config -siteLevelAggregationId 14
ipV6Address config -interfaceId 1
set addr [ipV6Address encode]
ixPuts $addr

ipV6Address decode $addr
ixPuts -nonewline [ipV6Address cget -prefixType]
ixPuts -nonewline ", "
ixPuts -nonewline [ipV6Address cget -topLevelAggregationId]
ixPuts -nonewline ", "
ixPuts -nonewline [ipV6Address cget -nextLevelAggregationId]
ixPuts -nonewline ", "
ixPuts -nonewline [ipV6Address cget -siteLevelAggregationId]
ixPuts -nonewline ", "
ixPuts [ipV6Address cget -interfaceId]

#Use the exposed members for the new RFC 3587 for configuring
ipV6Addr

package require IxTclHal

```

```

ipV6Address setDefault
ipV6Address config -prefixType ipv6GlobalUnicast3587
ipV6Address config -globalRoutingPrefix 42
ipV6Address config -subnetId 14
ipV6Address config -interfaceId 1
set addr [ipV6Address encode]
ixPuts $addr

ipV6Address decode $addr 1
ixPuts -nonewline [ipV6Address cget -prefixType]
ixPuts -nonewline ", "
ixPuts -nonewline [ipV6Address cget -globalRoutingPrefix]
ixPuts -nonewline ", "
ixPuts -nonewline [ipV6Address cget -subnetId]
ixPuts -nonewline ", "
ixPuts [ipV6Address cget -interfaceId]

#Set sourceAddrMode to
ipV6IncrementGlobalUnicastGlobalRoutingPrefixId(17)

sourceAddrModeipV6 setDefault
ipV6 config -trafficClass 3
ipV6 config -flowLabel 0
ipV6 config -hopLimit 255
ipV6 config -sourceAddr
"2444:4444:4444:5555:6666:6666:6666:6666"
ipV6 config -sourceMask 32
ipV6 config -sourceAddrMode 17
ipV6 config -sourceStepSize 1
ipV6 config -enableSourceSyncFromPpp false
ipV6 config -sourceAddrRepeatCount 10
ipV6 config -destAddr
"3555:5555:6666:6666:7777:7777:8888:8888"
ipV6 config -destMask 64
ipV6 config -destAddrMode ipV6Idle
ipV6 config -destStepSize 1
ipV6 config -nextHeader ipV6NoNextHeader
ipV6 config -enableDestSyncFromPpp false
ipV6 config -destAddrRepeatCount 10
ipV6 config -destGlobalUnicastMode 0
ipV6 config -sourceGlobalUnicastMode 1

//increment/decrement subnetid
-dest mask is fixed to 48 in this case; anything else will fail
package req IxTclHal
ixInitialize loopback
set chassis 1
set card 3
set port 1
set stream 1
stream get $chassis $card $port $stream
protocol setDefault
protocol config -ethernetType ethernetII
protocol config -name ipV6
ipV6Address config -prefixType ipv6GlobalUnicast3587
ipV6Address config -globalRoutingPrefix 42
ipV6Address config -subnetId 14
ipV6Address config -interfaceId 55
set addr [ipV6Address encode]
ipV6 config -destGlobalUnicastMode 1
ipV6 config -destMask 48
ipV6 config -destAddrMode 19

```

```

ipV6 config -destStepSize 1
ipV6 config -destAddr $addr
ipV6 set $chassis $card $port
stream set $chassis $card $port $stream
port write $chassis $card $port
-----
Second script for incr/decr subnet id:

package req IxTclHal
ixInitialize loopback
set chassis 1
set card 3
set port 1
set stream 1
stream get $chassis $card $port $stream
protocol setDefault
protocol config -ethernetType ethernetII
protocol config -name ipV6
ipV6Address config -prefixType ipV6GlobalUnicast3587
ipV6Address config -globalRoutingPrefix 42
ipV6Address config -subnetId 14
ipV6Address config -interfaceId 55
set addr [ipV6Address encode]
ipV6 config -destGlobalUnicastMode 1
ipV6 config -destMask 48
ipV6 config -destAddrMode ipV6IncrementSubnetId
ipV6 config -destStepSize 1
ipV6 config -destAddr $addr

ipV6 set $chassis $card $port
stream set $chassis $card $port $stream
port write $chassis $card $port
-----
package req IxTclHal
ixInitialize loopback
set chassis 1
set card 3
set port 1
set stream 1
stream get $chassis $card $port $stream
protocol setDefault
protocol config -ethernetType ethernetII
protocol config -name ipV6
ipV6Address config -prefixType ipV6GlobalUnicast3587
ipV6Address config -globalRoutingPrefix 42
ipV6Address config -subnetId 14
ipV6Address config -interfaceId 55
set addr [ipV6Address encode]
ipV6 config -destGlobalUnicastMode 1
ipV6 config -destMask 48
ipV6 config -destAddrMode ipV6DecrementSubnetId
ipV6 config -destStepSize 1
ipV6 config -destAddr $addr
ipV6 set $chassis $card $port
stream set $chassis $card $port $stream
port write $chassis $card $port

```

SEE ALSO

NAME - ipV6Authentication

ipV6Authentication - configure an IPv6 Authentication extension header

SYNOPSIS	ipV6Authentication <i>sub-command options</i>
-----------------	---

DESCRIPTION	The <i>ipV6Authentication</i> command creates an authentication extension header to be used in an <i>ipV6</i> header. This type of extension header is added to the <i>ipV6</i> header using <i>ipV6 addExtensionHeader</i> .
--------------------	---

STANDARD OPTIONS

authentication A variable length string containing the packets integrity check value (ICV).
(default = {00 00 00 00})

nextHeader *Read-only.* The type of the next extension header.

Option	Value	Usage
<i>ipV6HopByHopOptions</i>	0	Next header is hop-by-hop options.
<i>ipV6Routing</i>	43	Next header has routing options.
<i>ipV6Fragment</i>	44	Payload is a fragment.
<i>ipV6EncapsulatingSecurityPayload</i>	50	Next header is an IPSEC ESP.
<i>ipV6Authentiication</i>	51	Next header is an IPSEC AH.
<i>ipV6NoNextHeader</i>	59	<i>(default)</i> There is no next header.
<i>ipV6DestinationOptions</i>	60	Next header has destination options.
<i>ipV4ProtocolTcp</i>	6	Next header is TCP.
<i>ipV4ProtocolUdp</i>	17	Next header is UDP.
<i>icmpV6</i>	58	Next header is ICMP V6.
<i>ipV4ProtocolIpv4</i>		Next header is IPv4
<i>ipV4ProtocolTcp</i>		Next header is IPv4 with TCP
<i>ipV4ProtocolGre</i>		Next header is IPv4 with GRE
<i>ipV4ProtocolUdp</i>		Next header is IPv4 with UDP
<i>ipV4ProtocolIpv6icmp</i>		Next header is IPv4 with ICMP

payloadLength The length of the authentication data, expressed in 32-bit words. *(default = 2)*

reserved *Read-only.* Not currently used.

securityParam Index The security parameter index (SPI) associated with the authentication header.
(default = 0)

sequenceNumberField A sequence counter for the authentication header. *(default = 0)*

COMMANDS

The *ipV6Authentication* command is invoked with the following sub-commands.
If no sub-command is specified, returns a list of all sub-commands available.

ipV6Authentication *cget option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6Authentication* command.

ipV6Authentication *config option value*

Modify the IPv6 Authentication address table configuration options of the port. If no *option* is specified, returns a list describing all of the available *ipV6Authentication* options (see STANDARD OPTIONS).

ipV6Authentication *setDefault*

Sets default values for all IPv6 Authentication configuration options.

EXAMPLES

See examples under *ipV6*.

SEE ALSO

ipV6, ipV6Destination, ipV6Fragment, ipV6Routing, ipV6HopByHop

NAME - ipV6Destination

ipV6Destination - configures an IPv6 destination extension header

SYNOPSIS

ipV6Destination *sub-command options*

DESCRIPTION

The *ipV6Destination* command creates a destination extension header to be used in an *ipV6* header. This type of extension header is added to the *ipV6* header using *ipV6 addExtensionHeader*.

The destination extension header options must be configured separately, using the following commands:

Hop by Hop commands	Value	Description
<i>ipV6OptionPAD1</i>	0	The IPv6 PAD1 destination option.
<i>ipV6OptionPADN</i>	1	The IPv6 PADN destination option.
<i>ipV6OptionHomeAddress</i>	2	The IPv6 Home Address destination option.

STANDARD OPTIONS

headerExtLength

Read-only. The length of the header extension.

nextHeader

Read-only. The type of the next extension header.

Option	Value	Usage
<i>ipV6HopByHopOptions</i>	0	Next header is hop-by-hop options.
<i>ipV6Routing</i>	43	Next header has routing options.
<i>ipV6Fragment</i>	44	Payload is a fragment.
<i>ipV6EncapsulatingSecurityPayload</i>	50	Next header is an IPSEC ESP.
<i>ipV6Authentiication</i>	51	Next header is an IPSEC AH.
<i>ipV6NoNextHeader</i>	59	There is no next header.
<i>ipV6DestinationOptions</i>	60	Next header has destination options.
<i>tcp</i>	6	Next header is TCP.
<i>udp</i>	17	Next header is UDP.
<i>icmpV6</i>	58	Next header is ICMP V6.
<i>ipV4ProtocolIpv4</i>		Next header is IPv4
<i>ipV4ProtocolTcp</i>		Next header is IPv4 with TCP
<i>ipV4ProtocolGre</i>		Next header is IPv4 with GRE
<i>ipV4ProtocolUdp</i>		Next header is IPv4 with UDP
<i>ipV4ProtocolIpv6Icmp</i>		Next header is IPv4 with ICMP

COMMANDS

The *ipV6Destination* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ipV6Destination addOption *value*

Adds the specified option header to the packet.

ipV6Destination *cget* *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6Destination* command.

ipV6Destination *clearAllOptions*

Clears all options from the packet, with the exception of PADN.

ipV6Destination *config* *option value*

Modify the IPv6 destination address table configuration options of the port. If no *option* is specified, returns a list describing all of the available *ipV6Destination* options (see STANDARD OPTIONS).

ipV6Destination *delOption* *option*

Deletes the specified IPv6 destination option from the packet.

ipV6Destination *getFirstOption* *option*

Read-only. Gets the first IPv6 destination option configured in the packet.

ipV6Destination *getNextOption* *option*

Read-only. The type of the next IPv6 destination option.

ipV6Destination *setDefault*

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *ipV6*.

SEE ALSO

ipV6, *ipV6Authentication*, *ipV6Fragment*, *ipV6Routing*, *ipV6HopByHop*

NAME - ipV6Fragment

ipV6Fragment - configure an IPv6 fragment extension header

SYNOPSIS

ipV6Fragment *sub-command options*

DESCRIPTION

The *ipV6Fragment* command creates a fragment extension header to be used in an *ipV6* header. This type of extension header is added to the *ipV6* header using *ipV6 addExtensionHeader*.

Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in *atmHeader*. The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command is with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

STANDARD OPTIONS

enableFlag *true / false*

Indicates whether there are more fragments to be received (*true*) or this is the last fragment (*false*). (*default = true*)

fragmentOffset

A 13-bit value which is the offset for the data contained in this packet, relative to the start of the fragmentable part of the original packet, in 8-octet units. (*default = 100*)

identification

A 32-bit value that uniquely identifies the original packet which is to be fragmented. (*default = 0x11112222*)

nextHeader

Read-only. The type of the next extension header.

Option	Value	Usage
<i>ipV6HopByHopOptions</i>	0	Next header is hop-by-hop options.
<i>ipV6Routing</i>	43	Next header has routing options.
<i>ipV6Fragment</i>	44	Payload is a fragment.
<i>ipV6EncapsulatingSecurityPayload</i>	50	Next header is an IPSEC ESP.
<i>ipV6Authentication</i>	51	Next header is an IPSEC AH.
<i>ipV6NoNextHeader</i>	59	(<i>default</i>) There is no next header.
<i>ipV6DestinationOptions</i>	60	Next header has destination options.
<i>tcp</i>	6	Next header is TCP.
<i>ipV4ProtocolTcp</i>	6	Next header is IPv4 with TCP
<i>udp</i>	17	Next header is UDP.
<i>ipV4ProtocolUdp</i>	17	Next header is IPv4 with UDP
<i>icmpV6</i>	58	Next header is ICMP V6.
<i>ipV4ProtocolIpv6Icmp</i>	58	Next header is IPv4 with ICMP
<i>ipV4ProtocolGre</i>	47	Next header is IPv4 with GRE

res	2-bit reserved field. (<i>default</i> = 3)
reserved	8-bit reserved field. (<i>default</i> = 30)

COMMANDS

The *ipV6Fragment* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ipV6Fragment cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6Fragment* command.

ipV6Fragment config *option value*

Modify the IP address table configuration options of the port. If no *option* is specified, returns a list describing all of the available *ipV6Fragment* options (see STANDARD OPTIONS).

ipV6Fragment setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *ipV6*.

SEE ALSO

ipV6, *ipV6Authentication*, *ipV6Destination*, *ipV6Routing*, *ipV6HopByHop*

NAME - ipV6HopByHop

ipV6HopByHop - configure an IPv6 hop by hop extension header

SYNOPSIS

ipV6HopByHop *sub-command options*

DESCRIPTION

The *ipV6HopByHop* command creates a hop by hop extension header to be used in an *ipV6* header. This type of extension header is added to the *ipV6* header using *ipV6 addExtensionHeader*.

The hop by hop extension header options must be configured separately, using the following commands:

Hop by Hop commands	Value	Description
<i>ipV6OptionPAD1</i>	0	The IPv6 PAD1 Hop by Hop option.
<i>ipV6OptionPADN</i>	1	The IPv6 PADN Hop by Hop option.
<i>ipV6OptionJumbo</i>	194	The IPv6 Jumbo Hop by Hop option.
<i>ipV6OptionRouterAlert</i>	5	The IPv6 Router Alert Hop by Hop option.
<i>ipV6OptionBindingUpdate</i>	198	The IPv6 Binding Update Hop by Hop option.
<i>ipV6OptionBindingAck</i>	7	The IPv6 Binding ACK Hop by Hop option.
<i>ipV6OptionBindingRequest</i>	8	The IPv6 Binding Request Hop by Hop option.
<i>ipV6OptionMipV6UniqueIdSub</i>	2	The IPv6 Unique ID Sub Hop by Hop option.
<i>ipV6OptionMipV6AlternativeCoaSub</i>	4	The IPv6 Alternative COA Sub Hop by Hop option.
<i>ipV6OptionUserDefine</i>	112	The IPv6 PAD1 Hop by Hop option.

STANDARD OPTIONS

headerExtLength

Read-only. The length of this header, in bytes.

nextHeader

Read-only. The type of the next extension header.

Option	Value	Usage
<i>ipV6HopByHopOptions</i>	0	Next header is hop-by-hop options.
<i>ipV6Routing</i>	43	Next header has routing options.
<i>ipV6Fragment</i>	44	Payload is a fragment.
<i>ipV6EncapsulatingSecurityPayload</i>	50	Next header is an IPSEC ESP.
<i>ipV6Authentiication</i>	51	Next header is an IPSEC AH.
<i>ipV6NoNextHeader</i>	59	(default) There is no next header.
<i>ipV6DestinationOptions</i>	60	Next header has destination options.
<i>tcp</i>	6	Next header is TCP.
<i>ipV4Protocol/Tcp</i>	6	Next header is IPv4 with TCP
<i>udp</i>	17	Next header is UDP.

Option	Value	Usage
<i>ipV4ProtocolUdp</i>	17	Next header is IPv4 with UDP
<i>icmpV6</i>	58	Next header is ICMP V6.
<i>ipV4ProtocolIpv6Icmp</i>	58	Next header is IPv4 with ICMP
<i>ipV4ProtocolGre</i>	47	Next header is IPv4 with GRE

COMMANDS

The *ipV6HopByHop* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ipV6HopByHop addoption *option*

Adds the specified hop by hop option header to the packet.

Hop by Hop commands	Value	Description
<i>ipV6OptionPAD1</i>	0	The IPv6 PAD1 Hop by Hop option.
<i>ipV6OptionPADN</i>	1	The IPv6 PADN Hop by Hop option.
<i>ipV6OptionJumbo</i>	194	The IPv6 Jumbo Hop by Hop option.
<i>ipV6OptionRouterAlert</i>	5	The IPv6 Router Alert Hop by Hop option.
<i>ipV6OptionBindingUpdate</i>	198	The IPv6 Binding Update Hop by Hop option.
<i>ipV6OptionBindingAck</i>	7	The IPv6 Binding ACK Hop by Hop option.
<i>ipV6OptionBindingRequest</i>	8	The IPv6 Binding Request Hop by Hop option.
<i>ipV6OptionMlpV6UniqueIdSub</i>	2	The IPv6 Unique ID Sub Hop by Hop option.
<i>ipV6OptionMlpV6AlternativeCoaSub</i>	4	The IPv6 Alternative COA Sub Hop by Hop option.
<i>ipV6OptionUserDefine</i>	112	The IPv6 PAD1 Hop by Hop option.

ipV6HopByHop cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6Routing* command.

ipV6HopByHop config *option*

Modify the ipV6HopByHop configuration options of the port. If no *option* is specified, returns a list describing all of the available ipV6HopByHop options (see STANDARD OPTIONS) for port.

pV6HopByHop clearAllOptions

Clears all options from the packet, with the exception of PAD1.

ipV6HopByHop delOption *option*

Deletes the specified hop by hop option from the packet.

ipV6HopByHop getFirstOption *option*

Read-only. Gets the first hop by hop option configured in the packet.

ipV6HopByHop **getNextOption *option***

Read-only. The type of the next hop by hop option.

ipV6HopByHop **setDefault**

Sets default values for all hop by hop configuration options.

EXAMPLES

See examples under *ipV6*.

SEE ALSO

ipV6, ipV6Authentication, ipV6Destination, ipV6Fragment, ipV6Routing

NAME - ipV6OptionPAD1

ipV6OptionPAD1 - configure an IPv6 PAD1 destination extension header to IPv6

SYNOPSIS *ipV6OptionPAD1**sub-command options*

DESCRIPTION The *ipV6OptionPAD1* command adds a PAD1 header packet.

STANDARD OPTIONS

optionType *Read only*. Returns the value for the option.

COMMANDS

ipV6OptionPAD1 config option value

Configures the value of the specified PAD1 option.

ipV6OptionPAD1 cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6OptionPAD1* command.

EXAMPLES See examples under *ipV6*.

SEE ALSO *ipV6*, *ipV6Destination*, *ipV6HopByHop*

NAME - ipV6OptionPADN

ipV6OptionPADN - configure an IPv6 PADN header

SYNOPSIS ipV6OptionPADN *sub-command options*

DESCRIPTION The *ipV6OptionPADN* command adds a PADN to the IPv6 packet.

STANDARD OPTIONS

length The length of the header in bytes.

optionType *Read only*. Returns the value for the option.

value The value of the header data

COMMANDS

ipV6OptionPadN config *option value*

Configures the value of the specified option.

ipV6OptionPadN cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6OptionPADN* command.

ipV6OptionPadN setDefault

Sets default values for all PADN configuration options.

EXAMPLES See examples under *ipV6*.

SEE ALSO *ipV6, ipV6Destination, ipV6HopByHop*

NAME - *ipV6OptionJumbo*

ipV6OptionJumbo - configure an IPv6 Jumbo hop by hop header

SYNOPSIS	<i>ipV6OptionJumbo sub-command options</i>
DESCRIPTION	The <i>ipV6OptionJumbo</i> command adds a Jumbo hop by hop header to the IPv6 packet.

STANDARD OPTIONS

length	The length of the header in bytes.
payload	The payload for the header (that is, 11 11 11).
optionType	<i>Read only</i> . Returns the value for the option.

COMMANDS

ipV6OptionJumbo config option value

Configures the value of the specified option.

ipV6OptionJumbo cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6OptionJumbo* command.

ipV6OptionJumbo setDefault

Sets default values for all Jumbo configuration options.

EXAMPLES	See examples under <i>ipV6</i> .
SEE ALSO	<i>ipV6</i> , <i>ipV6HopByHop</i>

NAME - ipV6OptionRouterAlert

ipV6OptionRouterAlert - configure an IPv6 Router Alert hop by hop header

SYNOPSIS	ipV6OptionRouterAlert <i>sub-command options</i>
DESCRIPTION	The <i>ipV6OptionRouterAlert</i> command adds a Router Alert hop by hop header to the IPv6 packet.

STANDARD OPTIONS

length	The length of the header in bytes.
optionType	<i>Read only.</i> Returns the value for the option.
routerAlert type	Specifies the type of router alert to include with the packet. Choices are:

Option	Usage
<i>ipV6RouterAlertMLD</i>	MLD router alerts.
<i>ipV6RouterAlertRSVP</i>	RSVP router alerts
<i>ipV6RouterAlertActiveNet</i>	Active network router alerts.

COMMANDS

ipV6OptionRouterAlert config *option value*

Configures the value of the specified IPv6 Router Alert option.

ipV6OptionRouterAlert cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6OptionRouterAlert* command.

ipV6OptionRouterAlert **setDefault**

Sets default values for all IPv6 Router Alert configuration options.

EXAMPLES	See examples under ipV6 .
SEE ALSO	inV6 , inV6HopByHop

NAME - ipV6OptionBindingUpdate

ipV6OptionBindingUpdate - configure an IPv6 BindingUpdate hop by hop header

SYNOPSIS	ipV6OptionBindingUpdate <i>sub-command options</i>
-----------------	--

DESCRIPTION	The <i>ipV6OptionBindingUpdate</i> command adds a BindingUpdate hop by hop header to the IPv6 packet.
--------------------	---

STANDARD OPTIONS

enableAcknowledge <i>true / false</i>	This flag sets the Acknowledge (A) bit to indicate that the sending mobile node is requesting that a Binding Acknowledgement be sent by the receiving node when it gets the Binding Update. (<i>default = false</i>)
enableBroadcasting <i>true / false</i>	Enables the bi-casting flag for the Binding Update header. (<i>default = false</i>)
enableDuplicate <i>true / false</i>	This flag sets the Duplicate Address Detection (D) bit, to indicate that the sending node wants the receiving node to perform Duplicate Address Detection for the mobile node's home address in this binding. The H and A bits MUST also be set for this action to be performed. (<i>default = false</i>)
enableHome <i>true / false</i>	This flag sets the Home Registration (H) bit to indicate that the sending node wants the receiving node to act as its home agent. (<i>default = false</i>)
enableMap <i>true / false</i>	Enables the map flag for the Binding Update header. (<i>default = false</i>)
enableRouter <i>true / false</i>	This flag indicates if the binding cache entry is for a mobile node advertised as a router by this node, on the behalf of the mobile node, in proxy Neighbor Advertisements. (<i>default = false</i>)
length	The length of the header in bytes.
lifeTime <i>integer</i>	(32-bit integer) The number of seconds remaining for the Binding Cache entry. When the value reaches zero, the binding MUST be considered expired and the Binding Cache entry MUST be deleted for the mobile node.
optionType	<i>Read only.</i> Returns the value for the option.
prefixLength <i>integer</i>	If the H-bit is set, this is the length of the routing prefix for the home address
sequenceNumber <i>integer</i>	(16-bit number) The mobile node uses this number in the Binding Update. The receiving node uses the same number in its Binding Acknowledgement, for matching. The Sequence number in each Binding Update to one destination address must be greater than the last.

COMMANDS

ipV6OptionBindingUpdate **config *option value***

Configures the value of the specified IPv6 BindingUpdate option.

ipV6OptionBindingUpdate **cget *option***

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6OptionBindingUpdate* command.

ipV6OptionBindingUpdate **setDefault**

Sets default values for all IPv6 BindingUpdate configuration options.

EXAMPLES

See examples under *ipV6*.

SEE ALSO

ipV6, *ipV6HopByHop*

NAME - ipV6OptionBindingAck

ipV6BindingAck - configure an IPv6 BindingAck hop by hop header

SYNOPSIS	<i>ipV6OptionBindingAck sub-command options</i>
DESCRIPTION	The <i>ipV6OptionBindingAck</i> command adds a BindingACK hop by hop header to the IPv6 packet.
<hr/>	
STANDARD OPTIONS	
length	The length of the header in bytes.
lifeTime	(in seconds) The length of time that the receiving node retains the binding update entry for this mobile node in its binding cache.
optionType	<i>Read only</i> . Returns the value for the option.
refresh	(in seconds) The mobile node SHOULD send a new Binding Update at this interval, to refresh the binding. The receiving node (the node which sends the Binding ACK) determines the refresh interval.
sequenceNumber	This integer is copied from the received Binding Update into the corresponding Binding ACK message
status	(8 bit integer) This value indicates the disposition of the Binding Update: 0-127 = Binding Update was accepted. >= 128 = Binding Update was rejected.
<hr/>	
COMMANDS	
ipV6OptionBindingAck config option value	Configures the value of the specified IPv6 BindingAck option.
ipV6OptionBindingAck cget option	Returns the current value of the IPv6 BindingAck configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the <i>ipV6OptionBindingAck</i> command.
ipV6OptionBindingAck setDefault	Sets default values for all IPv6 BindingAck configuration options.
<hr/>	
EXAMPLES	See examples under <i>ipV6</i> .
<hr/>	
SEE ALSO	<i>ipV6</i> , <i>ipV6HopByHop</i>

NAME - ipV6OptionHomeAddress

ipV6OptionHomeAddress - configure an IPv6 HomeAdress header

SYNOPSIS

ipV6OptionHomeAddress *sub-command options*

DESCRIPTION

The *ipV6OptionHomeAddress* command adds a HomeAddress to the IPv6 packet.

STANDARD OPTIONS

address

The home address for the mobile node that is sending the packet. (*default* = *0:0:0:0:0:0:0:0*)

length

The length of the header in bytes.

optionType

Read only. Returns the value for the option.

COMMANDS

ipV6OptionHomeAddress config *option value*

Configures the value of the specified IPv6 HomeAdress option.

ipV6OptionHomeAddress cget *option*

Returns the current value of the IPv6 HomeAdress configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6OptionHomeAddress* command.

ipV6OptionHomeAddress setDefault

Sets default values for all IPv6 HomeAdress configuration options.

EXAMPLES

See examples under *ipV6*.

SEE ALSO

ipV6, ipV6Destination, ipV6HopByHop

NAME - ipV6OptionBindingRequest

ipV6OptionBindingRequest - configure an IPv6 BindingRequest hop by hop header

SYNOPSIS *ipV6OptionBindingRequest sub-command options*

DESCRIPTION The *ipV6OptionBindingRequest* command adds a BindingRequest hop by hop header to the IPv6 packet.

STANDARD OPTIONS

length The length of the header in bytes.

optionType *Read only*. Returns the value for the option.

COMMANDS

ipV6OptionBindingRequest config option value

Configures the value of the specified IPv6 BindingRequest option.

ipV6OptionBindingRequest cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6OptionBindingRequest* command.

ipV6OptionBindingRequest setDefault

Sets default values for all IPv6 BindingRequest configuration options.

EXAMPLES See examples under *ipV6*.

SEE ALSO *ipV6, ipV6HopByHop*

NAME - ipV6OptionMIPv6UniqueIdSub

ipV6OptionMIPV6UniqueIdSub - configure an IPv6 MIpV6UniqueIdSub hop by hop header

SYNOPSIS	<code>ipV6OptionMIPv6UniqueIdSub</code> <i>sub-command options</i>
DESCRIPTION	The <code>ipV6OptionMIPv6UniqueIdSub</code> command adds a MIPv6UniqueIdSub hop by hop header to the IPv6 packet.
STANDARD OPTIONS	
length	The length of the header in bytes.
optionType	<i>Read only.</i> Returns the value for the option.
subUniqueId	A unique ID for the binding request. (<i>default = 0</i>)
COMMANDS	
<code>ipV6OptionMIPv6UniqueIdSub config</code> <i>option value</i>	Configures the value of the specified Pv6 MIPv6UniqueIdSub option.
<code>ipV6OptionMIPv6UniqueIdSub cget</code> <i>option</i>	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the <code>ipV6OptionMIPv6UniqueIdSub</code> command.
<code>ipV6OptionMIPv6UniqueIdSub setDefault</code>	Sets default values for all Pv6 MIPv6UniqueIdSub configuration options.
EXAMPLES	See examples under <i>ipV6</i> .
SEE ALSO	<i>ipV6, ipV6HopByHop</i>

NAME - ipV6OptionMIpV6AlternativeCoaSub

ipV6OptionMIpV6AlternativeCoaSub - configure an IPv6 MIpV6AlternativeCoaSub hop by hop header

SYNOPSIS

ipV6OptionMIpV6AlternativeCoaSub *sub-command options*

DESCRIPTION

The *ipV6OptionMIpV6AlternativeCoaSub* command adds a MIpV6AlternativeCoaSub hop by hop header to the IPv6 packet.

STANDARD OPTIONS

address The IPv6 address. (*default = 0:0:0:0:0:0:0:0*)

length The length of the header in bytes.

optionType *Read only*. Returns the value for the option.

COMMANDS

ipV6OptionMIpV6AlternativeCoaSub config *option value*

Configures the value of the specified IPv6 MIpV6AlternativeCoaSub option.

ipV6OptionMIpV6AlternativeCoaSub cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6OptionIpV6AlternativeCoaSub* command.

ipV6OptionMIpV6AlternativeCoaSub setDefault

Sets default values for all IPv6 MIpV6AlternativeCoaSub configuration options.

EXAMPLES

See examples under *ipV6*.

SEE ALSO

ipV6, *ipV6HopByHop*

NAME - ipV6OptionUserDefine

ipV6OptionUserDefine - configure an IPv6 User Defined hop by hop header

SYNOPSIS	ipV6OptionUserDefine <i>sub-command options</i>
-----------------	---

DESCRIPTION	The <i>ipV6OptionUserDefine</i> command adds a user defined hop by hop header to the IPv6 packet.
--------------------	---

STANDARD OPTIONS

length	The length of the header in bytes.
---------------	------------------------------------

optionType	<i>Read only.</i> Returns the value for the option.
-------------------	---

value	A user-defined data value, in byte pairs (that is, 00 00 00 00).
--------------	--

COMMANDS

ipV6OptionUserDefine **config** *option value*

Configures the value of the specified IPv6 User Defined option.

ipV6OptionUserDefine **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6OptionUserDefine* command.

ipV6OptionUserDefine **setDefault**

Sets default values for all IPv6 User Defined configuration options.

EXAMPLES	See examples under <i>ipV6</i> .
-----------------	----------------------------------

SEE ALSO	<i>ipV6, ipV6HopByHop</i>
-----------------	---------------------------

NAME - ipV6Routing

ipV6Routing - configure an IPv6 routing extension header

SYNOPSIS	ipV6Routing <i>sub-command options</i>
-----------------	--

DESCRIPTION	The <i>ipV6Routing</i> command creates a routing extension header to be used in an <i>ipV6</i> header. This type of extension header is added to the <i>ipV6</i> header using <i>ipV6 addExtensionHeader</i> .
--------------------	--

STANDARD OPTIONS

headerExtLength *Read-only*. The length of this header, in bytes.

nextHeader *Read-only*. The type of the next extension header.

Option	Value	Usage
<i>ipV6HopByHopOptions</i>	0	Next header is hop-by-hop options.
<i>ipV6Routing</i>	43	Next header has routing options.
<i>ipV6Routing</i>	44	Payload is a routing.
<i>ipV6EncapsulatingSecurityPayload</i>	50	Next header is an IPSEC ESP.
<i>ipV6Authentication</i>	51	Next header is an IPSEC AH.
<i>ipV6NoNextHeader</i>	59	There is no next header.
<i>ipV6DestinationOptions</i>	60	Next header has destination options.
<i>tcp</i>	6	Next header is TCP.
<i>udp</i>	17	Next header is UDP.
<i>icmpV6</i>	58	Next header is ICMP V6.
<i>ipV4ProtocolIpv4</i>		Next header is IPv4
<i>ipV4ProtocolTcp</i>		Next header is IPv4 with TCP
<i>ipV4ProtocolGre</i>		Next header is IPv4 with GRE
<i>ipV4ProtocolUdp</i>		Next header is IPv4 with UDP
<i>ipV4ProtocolIpv6Icmp</i>		Next header is IPv4 with ICMP

nodeList A list of 128-bit IPv6 addresses, which may be constructed with the *ipV6Address* command. (*default = {}*)

reserved 32-bit reserved field. (*default = {00 00 00 00}*)

routingType *Read-only*. The routing type, always 0.

segmentsLeft *Read-only*. Only used if the *routing Type* is not recognized by this node. Always 0 in this release.

COMMANDS

The *ipV6Routing* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ipV6Routing cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *ipV6Routing* command.

ipV6Routing config *option value*

Modify the IPv6 routing extension address table configuration options of the port. If no *option* is specified, returns a list describing all of the available *ipV6Routing* options (see STANDARD OPTIONS).

ipV6Routing setDefault

Sets default values for all IPv6 routing extension configuration options.

EXAMPLES

See examples under *ipV6*.

SEE ALSO

ipV6, ipV6Authentication, ipV6Destination, ipV6Fragment, ipV6HopByHop

NAME - ipx

ipx - configure the IPX parameters for a port on a card on a chassis

SYNOPSIS	<i>ipx sub-command options</i>
-----------------	--------------------------------

DESCRIPTION	The ipx command is used to configure the IPX-specific information used when building IPX-type packets if the protocol config-name has been set to <i>ipx</i> . Note that <i>stream</i> get must be called before this command's get sub-command.
--------------------	--

STANDARD OPTIONS

destNetwork	The network number of the network to which the destination node belongs. (<i>default</i> = {00 00 00 00})
--------------------	---

destNetworkCounterMode	Specifies how the destination network address is incremented or decremented. Possible values include:
-------------------------------	---

Option	Value	Usage
<i>ipxIdle</i>	0	(<i>default</i>) no change to network address regardless of destNetworkRepeatCounter
<i>ipxIncrement</i>	1	increment the network address for as many destNetworkRepeatCounter specified
<i>ipxDecrement</i>	2	decrement the network address for as many destNetworkRepeatCounter specified
<i>ipxContIncrement</i>	3	Continuously increment the network address for each frame
<i>ipxContDecrement</i>	4	Continuously decrement the network address for each frame
<i>ipxCtrRandom</i>	5	Generate random destination network address for each frame

destNetworkMaskSelect	Selects the bits in the 32-bit destination network address that are to be masked by the value set by destNetworkMaskValue . (<i>default</i> = 00 00 00 00)
------------------------------	--

destNetworkMaskValue	Value of the masked bits selected by destNetworkMaskSelect in the destination network address. (<i>default</i> = FF FF FF FF)
-----------------------------	---

destNetworkRepeatCounter	Number of destination network addresses the stream is going to be transmitted to. (<i>default</i> = 1)
---------------------------------	---

destNode	The physical address of the destination node. (<i>default</i> = 00 00 00 00 00 00)
-----------------	---

destNodeCounterMode	Specifies how the destination node is incremented or decremented. Note: Setting the destNodeCounterMode other then <i>ipxIdle</i> takes over one of the available UDFs. Possible values include:
----------------------------	---

Option	Value	Usage
<i>ipxIdle</i>	0	(<i>default</i>) no change to node regardless of destNodeRepeatCounter

Option	Value	Usage
<i>ipxIncrement</i>	1	increment the node for as many destNodeRepeatCounter specified
<i>ipxDecrement</i>	2	decrement the node for as many destNodeRepeatCounter specified
<i>ipxContIncrement</i>	3	Continuously increment the node for each frame
<i>ipxContDecrement</i>	4	Continuously decrement the node for each frame
<i>ipxCtrRandom</i>	5	Generate random destination node for each frame

destNodeMaskSelect

Selects the bits in the 48-bit destination node address that are to be masked by the value set by **destNodeMaskValue**. (*default* = 00 00 00 00 00 00)

destNodeMaskValue

Value of the masked bits selected by **destNodeMaskSelect** in the destination node. (*default* = FF FFFF FF FF FF FF)

destNodeRepeat Counter

Number of destination nodes the stream is going to be transmitted to. (*default* = 1)

destSocket

The socket address of the packet's destination process. (*default* = 0x4000) Well defined addresses include:

Option	Value	Usage
socketNcp	1105–0x0451	
socketSap	1106–0x0452	
socketRipx	1107–0x0453	
socketNetBios	1109–0x0455	
socketDiagnostics	1110–0x0456	
socketSerialization	1111–0x0457	

destSocketCounter Mode

Specifies how the destination socket is incremented or decremented. Note: Setting the **destSocketCounterMode** other then ipxIdle takes over one of the available UDFs. Possible values include:

Option	Value	Usage
<i>ipxIdle</i>	0	(<i>default</i>) no change to socket regardless of destSocketRepeatCounter
<i>ipxIncrement</i>	1	increment the socket for as many destSocketRepeatCounter specified
<i>ipxDecrement</i>	2	decrement the socket for as many destSocketRepeatCounter specified
<i>ipxContIncrement</i>	3	Continuously increment the socket for each frame
<i>ipxContDecrement</i>	4	Continuously decrement the socket for each frame
<i>ipxCtrRandom</i>	5	Generate random destination socket for each frame

destSocketMaskSelect

Selects the bits in destination socket address that are to be masked by the value set by **destSocketMaskValue**. (*default* = 00 00)

destSocketMaskValue

Value of the masked bits selected by **destSocketMaskSelect** in the destination socket. (*default* = FF FF)

destSocketRepeatCounter Number of destination sockets the stream is going to be transmitted to. (*default = 1*)

length The length of the IPX header plus the length of the data. (*default = 0*)

lengthOverride true/false Allows to change the length in ipx header. (*default = false*)

packetType This field indicates the type of service offered or required by the packet. Possible values include:

<i>typeUnknown</i>	0–0x00	Used for all packets not classified by any other type.
<i>typeRoutingInfo</i>	1–0x01	Routing Information Packet.
<i>typeEcho</i>	2–0x02	Echo
<i>typeError</i>	3–0x03	Error
<i>typeIpx</i>	4–0x04	(<i>default</i>) Service Advertising Packet.
<i>typeSpx</i>	5–0x05	Used for sequenced packets.
<i>typeNcp</i>	17–0x11	Used for NetWare Core Protocol Packets.
<i>typeNetBIOS</i>	20–0x14	Used for Novell netBIOS.
<i>typeNdsNcp</i>	104–0x68	Used for NetWare Core Protocol Packets.

sourceNetwork The network number of the network to which the source node belongs. (*default = 00 00 00 00*)

sourceNetworkCounterMode Specifies how the source network address is incremented or decremented. Note: Setting the **sourceNetworkCounterMode** other then ipxIdle takes over one of the available UDFs. Possible values include:

Option	Value	Usage
<i>ipxIdle</i>	0	(<i>default</i>) no change to network address regardless of sourceNetworkRepeatCounter
<i>ipxIncrement</i>	1	increment the network address for as many sourceNetworkRepeatCounter specified
<i>ipxDecrement</i>	2	decrement the network address for as many sourceNetworkRepeatCounter specified
<i>ipxContIncrement</i>	3	Continuously increment the network address for each frame
<i>ipxContDecrement</i>	4	Continuously decrement the network address for each frame
<i>ipxCtrRandom</i>	5	Generate random source network address for each frame

sourceNetworkMaskSelect Selects the bits in the 32-bit source network address that are to be masked by the value set by **sourceNetworkMaskValue**. (*default = 00 00 00 00*)

sourceNetworkMaskValue Value of the masked bits selected by **sourceNetworkMaskSelect** in the source network address. (*default = FF FF FF FF*)

sourceNetworkRepeatCounter Number of source network addresses the stream is going to be transmitted to. (*default = 1*)

sourceNode

The physical address of the source node. (*default* = 00 00 00 00 00 00)

sourceNodeCounterMode

Specifies how the source node is incremented or decremented. Note: Setting the **sourceNodeCounterMode** other then ipxIdle takes over one of the available UDFs. Possible values include:

Option	Value	Usage
<i>ipxIdle</i>	0	(<i>default</i>) no change to node regardless of sourceNodeRepeatCounter
<i>ipxIncrement</i>	1	increment the node for as many sourceNodeRepeatCounter specified
<i>ipxDecrement</i>	2	decrement the node for as many sourceNodeRepeatCounter specified
<i>ipxContIncrement</i>	3	Continuously increment the node for each frame
<i>ipxContDecrement</i>	4	Continuously decrement the node for each frame
<i>ipxCtrRandom</i>	5	Generate random source node for each frame

sourceNodeMaskSelect

Selects the bits in the 48-bit source node address that are to be masked by the value set by **sourceNodeMaskValue**. (*default* = 00 00 00 00 00 00)

sourceNodeMaskValue

Value of the masked bits selected by **sourceNodeMaskSelect** in the source node. (*default* = FF FF FF FF FF FF)

sourceNodeRepeatCounter

Number of source nodes the stream is going to be transmitted to. (*default* = 1)

sourceSocket

The socket address of the packet's source process. (*default* = 0x4000) Well known addresses include:

Option	Value	Usage
socketNcp	1105–0x0451	
socketSap	1106–0x0452	
socketRipx	1107–0x0453	
socketNetBios	1109–0x0455	
socketDiagnostics	1110–0x0456	
socketSerialization	1111–0x0457	

sourceSocketCounterMode

Specifies how the source socket is incremented or decremented. Note: Setting the **sourceSocketCounterMode** other then ipxIdle takes over one of the available UDFs. Possible values include:

Option	Value	Usage
<i>ipxIdle</i>	0	(<i>default</i>) no change to socket regardless of sourceSocketRepeatCounter
<i>ipxIncrement</i>	1	increment the socket for as many sourceSocketRepeatCounter specified
<i>ipxDecrement</i>	2	decrement the socket for as many sourceSocketRepeatCounter specified
<i>ipxContIncrement</i>	3	Continuously increment the socket for each frame sourceSocketRepeatCounter specified
<i>ipxContDecrement</i>	4	Continuously decrement the socket for each frame
<i>ipxCtrRandom</i>	5	Generate random source socket for each frame

sourceSocketMaskSelect Select
Selects the bits in source socket address that are to be masked by the value set by **sourceSocketMaskValue**. (*default = 00 00*)

sourceSocketMaskValue
Value of the masked bits selected by **sourceSocketMaskSelect** in the source socket. (*default = FF FF*)

sourceSocketRepeatCounter
Number of source sockets the stream is going to be transmitted to. (*default = 1*)

svrClientType
This allows the port to act either as a NetWare server or client. If set to server, then the port may send out SAP broadcasts to announce itself. Possible values include:

Option	Value	Usage
server	1	
client	2	(<i>default</i>)

transportControl
The number of routers that the packet has passed through. (*default = 0*)

COMMANDS
The **ipx** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ipx cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ipx** command.

ipx config option value

Modify the IPX configuration options of the port. If no *option* is specified, returns a list describing all of the available IPX options (see STANDARD OPTIONS) for port.

ipx decode capFrame [chassisID cardID portID]

Decodes a captured frame in the capture buffer and updates TclHal. **ipx cget option** command can be used after decoding to get the option data. Specific errors are:

- No connection to a chassis
- Invalid port number
- The captured frame is not a valid IPX frame

ipx get chassisID cardID portID

Gets the current configuration of the **ipx** object for port with id *portID* on card *cardID*, chassis *chassisID* from its hardware and sets the **ipx** class members with the current data. Note that *stream* get must be called before this command's get sub-command. Specific errors are:

- No connection to a chassis
- Invalid port number

ipx set *chasID cardID portID*

Sets the IPX configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **ipx config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

ipx setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

stream setDefault

protocol setDefault
protocol config -name ipx
protocol config -ethernetType ethernetII

ipx setDefault

ipx config -destNetwork {00 00 00 02}
```

```
ipx config -destNetworkCounterMode ipxIdle
ipx config -sourceNetwork {00 00 00 01}
ipx config -sourceNetworkCounterMode ipxIdle

ipx config -destNode {00 00 00 01 00 00}
ipx config -destNodeRepeatCounter 16
ipx config -destNodeCounterMode ipxDerecrement
ipx config -sourceNode {00 00 00 00 00 00}
ipx config -sourceNodeRepeatCounter 16
ipx config -sourceNodeCounterMode ipxIncrement

ipx config -destSocket 5
ipx config -sourceSocket 4
ipx set $chas $card $port

stream set $chas $card $port 1

ixWriteConfigToHardware portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

stream, protocol, ip, udp

NAME - *isl*

isl - configure the Cisco Inter-Switch Link (ISL) parameters for a port on a card on a chassis

SYNOPSIS

isl sub-command options

DESCRIPTION

The *isl* command is used to configure the ISL-specific information used when building ISL-type packets. This is enabled using **protocol config -enableISLtag true**. The encapsulated frame's Source and Destination MAC addresses are configured through the *stream config -da* and *-sa* commands. See *stream* on page A-747. The previously documented options to the *isl* command *encapDA* and *encapSA*, should not be used to set the MAC addresses but may be used to view the values.

STANDARD OPTIONS

bpdus Set for all Bridge Protocol Data Units that are encapsulated by the ISL packet. (*default = 0*)

encapDA *Read-only*. This value is set through the use of *stream config -da*.

encapSA *Read-only*. This value is set through the use of *stream config -sa*.

frameType The type field indicates the type of frame that is encapsulated. Options include:

Option	Value	Usage
islFrameEthernet	0	(<i>default</i>)
islFrameTokenRing	1	
islFrameFDDI	2	
islFrameATM	3	

index Value of the selected register. (*default = 0*)

islDA The address is a multicast address whose value in the first 40 bits of the DA indicate to the receiver that the packet is in ISL format. (*default = {01 00 0C 00 00}*)

islSA The source MAC address. The upper 3 bytes of this field are reflected in the hsa field. (*default = {00 00 0C 00 00 00}*)

length *Read-Only*. The calculated length of the ISL header.

reserved The *reserved* field of the ISL header. (*default = {00 00}*)

userPriority The low order two bits of this field indicate the priority of the packet as it passes through the switch. Priorities 0 to 7 are valid. (*default = 0*)

vlanID The Virtual LAN Identifier. (*default = 1*)

COMMANDS

The **isl** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

isl cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **isl** command.

isl config option value

Modify the ISL configuration options of the port. If no *option* is specified, returns a list describing all of the available ISL options (see STANDARD OPTIONS) for port.

isl decode capFrame [chassisID cardID portID]

Decodes a captured frame in the capture buffer and updates TclHal. **isl cget option** command can be used after decoding to get the option data. Specific errors are:

- No connection to a chassis
- Invalid port number
- The captured frame is not a valid ISL frame

isl get chassisID cardID portID

Gets the current ISL configuration of the port with id *portID* on card *cardID*, chassis *chassisID*. Call this command before calling **isl cget option** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

isl set chassisID cardID portID

Sets the ISL configuration of the port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **isl config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

isl setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
```

```
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
set card 1
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

stream setDefault
protocol setDefault
protocol config -name          ipV4
protocol config -ethernetType ethernetII
protocol config -enableISLtag true

isl setDefault
isl config -vlanID           42
isl set $chas $card $port

stream config -sa             {01 02 03 04 05 06}
stream config -da             {02 03 04 05 06 07}
stream set $chas $card $port 1

ixWriteConfigToHardware portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

protocol, stream

NAME - lasi

lasi - configure the link alarm status interrupt settings for XENPAK modules

SYNOPSIS	<i>lasi sub-command options</i>
DESCRIPTION	The <i>lasi</i> command is used to configure the OUI address and interrupt settings associated with XENPAK modules. The OUI (Organizationally Unique Identifier) device address <i>ouiDeviceAddress</i> allows communications with the XENPAK device registers that control the conditions under which an alarm interrupt occurs. The particular conditions are controlled by the <i>rxAlarmControlRegister</i> , <i>txAlarmControlRegister</i> and <i>controlRegister</i> . The particular values in these control registers is covered in the <i>XENPAK 10 GIGabit Ethernet MSA, Issue 3.0</i> .
<hr/>	
STANDARD OPTIONS	
controlRegister	The value for the control register. (<i>default = "00 00"</i>)
enableAutoDetectedOUIDeviceAddress <i>enable / disable</i>	Enables the ability of the port to automatically detect the OUI device address. (<i>default = disable</i>)
enableMonitoring <i>true / false</i>	Enables active monitoring of the LASI status registers so as to clear the interrupt signal. (<i>default = false</i>)
ouiDeviceAddress	The OUI device address for the LASI registers. (<i>default = 3</i>)
rxAlarmControlRegister	The receive alarm register contents. (<i>default = "00 00"</i>)
txAlarmControlRegister	The transmit alarm register contents. (<i>default = "00 00"</i>)
<hr/>	
COMMANDS	The lasi command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
lasi cget option	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the lasi command.
lasi config option value	Modify the lasi configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available lasi options (see STANDARD OPTIONS) for port.
lasi get chasID cardID portID	Gets the current lasi configuration of the port with id <i>portID</i> on card <i>cardID</i> , chassis <i>chasID</i> . Call this command before calling lasi cget option to get the value of the configuration option. Specific errors are:
	<ul style="list-style-type: none"> • No connection to a chassis • Invalid port number

lasi set *chasID cardID portID*

Sets the lasi configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **lasi config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

lasi setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chId [ixGetChassisID $host]

set cardId 60
set portId 1
set portList [list [list $chId $cardId $portId]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

set retCode "PASS"

lasi setDefault

lasi config -ouiDeviceAddress      1
lasi config -rxAlarmControlRegister {ff ff}
lasi config -txAlarmControlRegister 0x55
lasi config -controlRegister       0xffff
```

```
if {[lasi set $chId $cardId $portId]} {  
    ixPuts $::ixErrorInfo  
    set retCode "FAIL"  
    break  
}  
  
# Let go of the ports that we reserved  
ixClearOwnership $portList  
# Disconnect from the chassis we're using  
ixDisconnectFromChassis $host  
# If we're running on UNIX, disconnect from the TCL Server  
if [isUNIX] {  
    ixDisconnectTclServer $host  
}  
  
return $retCode
```

SEE ALSO*mii*

NAME - latencyBin

latencyBin - retrieve statistics associated with a latency bin of a packet group.

SYNOPSIS

latencyBin *sub-command options*

DESCRIPTION

The *latencyBin* command is used to retrieve the statistics associated with a particular latency bin in a packet groups, such as minimum latency, maximum latency and average latency.

The latency bin information must be retrieved through calls to *packetGroupStats*, *getFirstLatencyBin*, *getNextLatencyBin* and *getLatencyBin*.

STANDARD OPTIONS**bitRate**

Read-only. 64-bit value. The bit rate for the frames.

byteRate

Read-only. 64-bit value. The byte rate for the frames.

firstTimeStamp

Read-only. 64-bit value. The time stamp of the first packet received.

frameRate

Read-only. 64-bit value. The frame rate for the frames.

lastTimeStamp

Read-only. 64-bit value. The time stamp of the last packet received.

maxLatency

Read-only. 64-bit value. Maximum latency of all frames of this packet group.

minLatency

Read-only. 64-bit value. Minimum latency of all frames of this packet group.

numFrames

Read-only. 64-bit value. Total number of frames in this latency bin.

startTime

Read-only. Floating point value. The start time of the latency bin, expressed in microseconds.

stopTime

Read-only. Floating point value. The stop time of the latency bin, expressed in microseconds.

COMMANDS

The *latencyBin* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

latencyBin cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *latencyBin* command.

EXAMPLES

See examples under *packetGroup*.

SEE ALSO

packetGroup, *packetGroupStats*, *stream*

NAME - lcas

lcas - sets up LCAS configuration for a circuit to receive and/or transmit.

SYNOPSIS	lcas <i>sub-command options</i>
DESCRIPTION	The <i>lcas</i> command is used to set up LCAS configuration for receive and/or transmit. This enables configuring the LCAS debug/trace messages.
STANDARD OPTIONS	
rsAck	Configure the timeout value for Rs_Ack(s) for Rx Lcas. (<i>default = 10</i>)
holdOff	Configure the hold off timeout for Rx Lcas. (<i>default = 10</i>)
waitToRestore	Configure the wait to restore timeout for the Rx Lcas. (<i>default = 10</i>)
COMMANDS	
	The lcas command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
lcas cget option	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the lcas command.
lcas config option value	Modify the configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS).
lcas get chassisID cardID portID circuitID	Gets the existing LCAS configuration for the circuit with the given circuit ID. Return values: <ul style="list-style-type: none">• 0–OK• 1–Tcl error• 100–Port unavailable• 101–Unsupported feature
lcas set chassisID cardID portID circuitID	Modify the existing LCAS configuration for the circuit with the given circuit ID. Return values: <ul style="list-style-type: none">• 0–OK• 1–Tcl error• 100–Port unavailable• 101–Unsupported feature
lcas setDefault	Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example on page A-667.

SEE ALSO

sonetCircuit, sonetCircuitList, sonetCircuitProperties

NAME - linkFaultSignaling

linkFaultSignaling - configure and start/stop link fault signalling

SYNOPSIS

linkFaultSignaling *sub-command options*

DESCRIPTION

The *linkFaultSignaling* command is used to define a series or continuous stream of link fault signals. The series/stream consists of good and bad period, where the bad periods may send local, remote or custom errors. Errors are called ordered sets; two, named A and B, are available for insertion.

STANDARD OPTIONS

contiguousErrorBlocks

The number of contiguous errored blocks to insert at a time. This must be an even number between 2 and 30. The type of error block inserted is determined by the setting of the *sendSetsMode* option. (*default = 2*)

contiguousGoodBlocks

The number of contiguous non-errored blocks to insert at a time. This must be an even number between 2 and 512. (*default = 2*)

enableLoopContinuously true / false

If true, the cycle of errored and non-errorored blocks is applied continuously. Errors are inserted when the *startErrorInsertion* sub-command is called and stopped when the *stopErrorInsertion* sub-command is called. (*default = true*)

enableTxIgnoresRx LinkFault true / false

If true, then the port continues to transmit even when the port has received a remote link fault. (*default = false*)

loopCount

If *enableLoopContinuously* is false, then this is the number of times that good-bad cycles is applied. The setting of the *sendSetsMode* option determines whether there are one or two good-bad cycles per loop. (*default = 0*)

orderedSetTypeA

Determines the type of ordered set to be used for type A errors.

Option	Value	Usage
linkFaultLocal	0	(<i>default</i>) A local fault.
linkFaultRemote	1	A remote fault.
linkFaultCustom	2	A custom fault, specified through the use of the <i>customOrderedSet</i> command.

orderedSetTypeB

Determines the type of ordered set to be used for type B errors.

Option	Value	Usage
linkFaultLocal	0	A local fault.
linkFaultRemote	1	(<i>default</i>) A remote fault.
linkFaultCustom	2	A custom fault, specified through the use of the <i>customOrderedSet</i> command.

sendSetsMode

Indicates whether to transmit alternating good-bad blocks using only Type A blocks, only Type B blocks or alternating between them. The choices are:

Option	Value	Usage
linkFaultSendTypeA	0	Use type A ordered sets only.
linkFaultSendTypeB	1	Use type B ordered sets only.
linkFaultCustom	2	(default) Use type A ordered sets, then good blocks, type B ordered sets and then good blocks. Each cycle forms one loop count as used in <i>loopCount</i> .

COMMANDS

The **linkFaultSignaling** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

linkFaultSignaling cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **linkFaultSignaling** command.

linkFaultSignaling config option value

Modify the linkFaultSignaling configuration options of the port. If no *option* is specified, returns a list describing all of the available linkFaultSignaling options (see STANDARD OPTIONS) for port.

linkFaultSignaling get chasID cardID portID

Gets the current linkFaultSignaling configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Call this command before calling **linkFaultSignaling cget option** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

linkFaultSignaling set chasID cardID portID

Sets the linkFaultSignaling configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **linkFaultSignaling config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

linkFaultSignaling startErrorInsertion chasID cardID portID

Starts the process of error insertion on the indicated port. Specific errors are:

-

linkFaultSignaling setDefault

Sets to IxTclHal default values for all configuration options.

linkFaultSignaling stopErrorInsertion chasID cardID portID

Stops the process of error insertion on the indicated port. This can be used to stop error insertion when *enableLoopContinuously* is true, or to prematurely stop error insertion when *loopCount* is used. Specific errors are:

•

EXAMPLES

```

package require IxTclHal

set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 55
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get current link fault signaling data
if [linkFaultSignaling get $chas $card $port] {
    ixPuts "Error in linkFaultSignaling"
}

# Define a custom ordered set A
# This is actually equivalent to a local fault
customOrderedSet config -blockType 0x4B
customOrderedSet config -syncBits 0x02
customOrderedSet config -byte1 0x00
customOrderedSet config -byte2 0x00
customOrderedSet config -byte3 0x01
customOrderedSet config -byte4 0x00
customOrderedSet config -byte5 0x00
customOrderedSet config -byte6 0x00
customOrderedSet config -byte7 0x00
if [customOrderedSet set linkFaultOrderedSetTypeA] {
    ixPuts "Error in customOrderedSet set"
}

# Set up link fault signalling, continuous insertion
# of 14 errors, 200 good
linkFaultSignaling config -sendSetsMode
linkFaultAlternateOrderedSets
linkFaultSignaling config -contiguousErrorBlocks 14
linkFaultSignaling config -contiguousGoodBlocks 200

```

```
linkFaultSignaling config -enableLoopContinuously true
linkFaultSignaling config -orderedSetTypeA           linkFaultCustom
linkFaultSignaling config -orderedSetTypeB          linkFaultRemote

if [linkFaultSignaling set $chassis $card $port] {
    ixPuts "Error in linkFaultSignaling set"
}

ixWriteConfigToHardware portList

if [linkFaultSignaling startErrorInsertion $chassis $card $port] {
    ixPuts "Error in linkFaultSignaling startErrorInsertion"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*customOrderedSet*

NAME - macSecChannel

macSecChannel - configure and hold MacSec channel information

SYNOPSIS macSecChannel *sub-command options*

DESCRIPTION The **macSecChannel** command is used to hold and configure the MacSec channel information for each direction.

STANDARD OPTIONS

channelName	Allows configuration of the MacSec channel name.
macAddress	Allows configuration of the MacSec channel MAC address. (<i>default</i> = '00 00 00 00 00 00')
portIdentifier	<i>Read only</i> . Displays the port identifier information. (<i>default</i> = 0)
enableAssociation	Enables/disables the secure association number. (<i>default</i> = <i>false</i>)
associationKey	Used to configure the key for secure association number. (<i>default</i> = '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00')
associationNumber	<i>Read only</i> . Displays the current secure association number. (<i>default</i> = 0 = <i>secureAN0</i>)
direction	<i>Read only</i> . Displays the current channel direction if it is a macSecTx or macSecRx. (<i>default</i> = <i>macSecTransmit</i>)

COMMANDS

The **macSecChannel** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

macSecChannel **getAssociation** *secureAssociationNumber*

Gets all the configurations for the given secure association with the given *secureAssociationNumber*.

The supported options for *secureAssociationNumber* are:

Option
secureAN0
secureAN1
secureAN2
secureAN3

macSecChannel **setAssociation** *secureAssociationNumber*

Sets all the configurations for the given secure association with the given *secureAssociationNumber*.

The supported options for secureAssociationNumber are:

Option
secureAN0
secureAN1
secureAN2
secureAN3

macSecChannel `setDefault`

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See Examples under *macSecTag*.

SEE ALSO

macSecTx, *macSecRx*, *macSecTag*

NAME - macSecRx

macSecRx - configures the basic MacSec receive parameters of the port.

SYNOPSIS	macSecRx <i>sub-command options</i>
DESCRIPTION	The macSecRx command is used to configure the basic MacSec receive parameters of the port.
STANDARD OPTIONS	
numChannels	<i>Read only.</i> Displays the number of secure Rx channels. (<i>default = 0</i>)
confidentialityOffset	Used to configure the MacSec port confidentiality offset. (<i>default = 0</i>) Valid choices are: 0, 4, 30, 50.
COMMANDS	
	The macSecRx command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
macSecRx setDefault	Sets to IxTclHal default values for all configuration options.
macSecRx select <i>chassisID cardID portID</i>	Selects the port to set or retrieve data from. By default, it fills in the object with the receive configuration details.
macSecRx set	Sets the MacSecRx configuraton from IxTclHal to local IxHal object.
macSecRx get	Gets the MacSecRx configuraton from local IxHal to IxTclHal.
macSecRx addChannel	Adds the configured Connectivity Association channel configuration data for this port into the IxHal.
macSecRx getChannel <i>secureChannelId</i>	Retrieves the configured Connectivity Association Rx channel configuration data for the specified ID on this port from the IxHal.
macSecRx setChannel <i>secureChannelId</i>	Sets the corresponding configured Connectivity Association Rx channel configuration data for the specified ID on this port to the IxHal.

macSecRx delChannel *secureChannelId*

Deletes the specified configured Connectivity Association Rx channel configuration data for the specified ID on this port from the IxHal.

macSecRx clearAllChannels

Deletes all the configured Connectivity Association channels for the selected port from the IxHal.

macSecRx getFirstChannel

Retrieves the first configured Connectivity Association channel configuration data for the selected port from the IxHal.

macSecRx getNextChannel

Retrieves the next configured Connectivity Association channel configuration data for the selected port from the IxHal.

EXAMPLES

See Examples under *macSecTag*.

SEE ALSO

macSecChannel, *macSecTx*, *macSecTag*

NAME - macSecTag

macSecTag - contains the MacSec header.

SYNOPSIS	macSecTag <i>sub-command options</i>
DESCRIPTION	The macSecTag command is used to contain the MacSec header. This is the per-stream configuration. Note that <i>macSecTag</i> needs to be configured before <i>stream</i> is set.
STANDARD OPTIONS	
tciVersion	Allows the configuration of version. (<i>default = 0</i>)
enableTciVersionOverride	Allows the enabling of version override. (<i>default = false</i>)
enableForceByteCorruption	Allows the enabling of forced byte corruption. (<i>default = false</i>)
enableOverrideFlagRestriction	Allows the enabling of tag control information override. (<i>default = false</i>)
enableTciEndStation	Allows the enabling of tag control information end station override. (<i>default = false</i>) If this parameter is set to <i>true</i> along with <i>enableTciIncludeSci</i> when <i>enableOverrideFlagRestriction</i> is not enabled, then it is an invalid configuration and the set command fails with the error message posted in the <i>TclEvents.log</i> . Only one can be enabled at a time, either <i>enableTciEndStation</i> or <i>enableTciIncludeSci</i> .
enableTciIncludeSci	Allows the enabling of tag control information include Sci. (<i>default = false</i>) If this parameter is set to <i>true</i> along with <i>enableTciEndStation</i> when <i>enableOverrideFlagRestriction</i> is not enabled, then it is an invalid configuration and the set command fails with the error message posted in the <i>TclEvents.log</i> . Only one can be enabled at a time, either <i>enableTciEndStation</i> or <i>enableTciIncludeSci</i> .
enableTciSingleCopyBroadcast	Allows the enabling of tag control information single copy broadcast. This parameter cannot be enabled if <i>enableTciIncludeSci</i> is enabled. (<i>default = false</i>)
enableTciEncryption	Allows the enabling of tag control information encryption. (<i>default = false</i>)
enableTciChangedText	Allows the enabling of tag control information changed text. (<i>default = false</i>)
associationNumber	Allows the configuration of association number. (<i>default = 0</i>)
macAddress	Allows the configuration of MAC address when <i>enableTciIncludeSci = true</i> . (<i>default = '00 00 00 00 00 00'</i>)

portIdentifier	Allows the configuration of the port identifier value when enableTciIncludeSci = true. (<i>default = 0</i>)
enableShortLengthOverride	Allows the enabling of short length override. (<i>default = false</i>)
shortLength	Allows the configuration of short length. (<i>default = 0</i>)
packetNumber	Allows the configuration of packet number. (<i>default = '00 00 00 00'</i>)

COMMANDS

The **macSecTag** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

macSecTag setDefault

Sets to IxTclHal default values for all configuration options.

macSecTag set *chassisID cardID portID*

Commits to IxHAL the macSecTag header for a particular portID.

macSecTag get *chassisID cardID portID*

Retrieves from IxHAL the macSecTag header for a particular portID.

macSecTag decode *capFrame chassisID cardID portID*

Decodes the MacSec Tag Frame and populates the TCLMacSecTag object if the feature is supported and decoding was successful.

EXAMPLES

```
package req IxTclHal
set hostname loopback
ixConnectToChassis $astro
set retCode "PASS"

if { [ixConnectToChassis $hostName] } {
    errorMsg "Error connecting to $hostName"
    set retCode "FAIL"
}
set chassisId[chassis cget -id]
set portList[list]

for { set cardId 1 } {$cardId <= [chassis cget -maxCardCount]}
{incr cardId} {
    if {[card get $chassisId $cardId] == $::TCL_OK} {
        set portId 1
        if {[port isValidFeature $chassisId $cardId $portId
$::portFeatureMACSec]} {
            port setModeDefaults $chassisId $cardId $portId
            lappend portList [list $chassisId $cardId $portId]
        }
    }
}
if {[llength $portList] == 0} {
    errorMsg "No ports in port list that support MACSec"
    set retCode "FAIL"
    return $retCode
}
```

```

foreach port $portList {
    scan $port "%d %d %d" chassisId cardId portId

    if {[ macSecTx select $chassisId $cardId $portId $streamId] } {
        errorMsg "Error setTx macSec: "
        set retCode "FAIL"
        break
    }
    if {[ macSecRx select $chassisId $cardId $portId $streamId] } {
        errorMsg "Error setTx macSec: "
        set retCode "FAIL"
        break
    }

    macSecChannel setDefault
    macSecChannel config -enable true
    macSecChannel config -key "aa de bb 11 42"
    if {[macSecChannel setAssociation $::secureAN0]} {
        errorMsg "Error setting macSecChannel on secureAN0"
        set retCode "FAIL"
        break
    }

    macSecChannel config -key "00 ig 11 00 20"
    if {[macSecChannel setAssociation $::secureAN1]} {
        errorMsg "Error setting macSecChannel on secureAN1"
        set retCode "FAIL"
        break
    }

    # Add the first secure channel
    if {[ macSecTx addChannel]} {
        errorMsg "Error adding Tx macSec connectivity association "
        set retCode "FAIL"
        break
    }

    # Add the second secure channel
    if {[ macSecRx addChannel]} {
        errorMsg "Error adding Rx macSec connectivity association "
        set retCode "FAIL"
        break
    }
    set macSecChannelId1
    if {[ macSecTx get $macSecChannelId} {
        errorMsg "Error getting macSecTx: "
        set retCode "FAIL"
        break
    }

    ixPuts "Number of Tx secure channels:[macSecTx cget -
numChannels]"

    if {[ macSecRx getChannel $macSecChannelId} {
        errorMsg "Error getting macSecRx: "
        set retCode "FAIL"
        break
    }

    ixPuts "Number of Rx secure channels:[macSecRx cget -
numChannels]"

```

```

if {[ macSecTx getFirstChannel ]} {
    errorMsg "Error adding macSec: "
    set retCode "FAIL"
    break
}
if {[macSecChannel getAssociation $::secureAN1]} {
    errorMsg "Error setting macSecChannel on secureAN1"
    set retCode "FAIL"
    break
}
    ixPuts "association number [macSecChannel cget - associationNumber]"
    ixPuts "association key [macSecChannel cget - associationKey]

if {[ macSecRx getNextChannel ]} {
    errorMsg "Error adding macSec: "
    set retCode "FAIL"
    break
}
    ixPuts "Number of Rx secure channels:[macSecRx cget - numChannels]"
}

ixWritePortsToHardware portList
ixCheckLinkState portList

stream setDefault
protocol setDefault
protocol config -enableMacSec$::true

foreach port $portList {
    set streamed 1
    stream setDefault
    stream config -name "my MACSec stream"
    if [stream set $chassisId $cardId $portId $streamId] {
        errorMsg "Error setting stream on port
$chassisId.$cardId.$portId $streamId"
        set retCode "FAIL"
        break
    }
    macSecTag setDefault
    macSecTag config - enableOverrideFlagRestriction true
    macSecTag config -enableTciEndStation true
    macSecTag config -enableTciIncludeSci true
    macSecTag config -macAddress "00 11 22 33 44 56"
    macSecTag config -portIdentifier 42
    macSecTag config -associationNumber secureAN1
    if {[macSecTag set $chassisId $cardId $portId $streamId] {
        errorMsg "Error setting macSecTag header on
$chassisId.$cardId.$portId $streamId"
        set retCode "FAIL"
        break
    }
}

ixWriteConfigToHardware portList

```

SEE ALSO*macSecChannel, macSecTx, macSecRx*

NAME - macSecTx

macSecTx - configures the basic MacSec transmit parameters of the port.

SYNOPSIS	macSecTx <i>sub-command options</i>
DESCRIPTION	The macSecTx command is used to configure the basic MacSec transmit parameters of the port.

STANDARD OPTIONS

numChannels	<i>Read only.</i> Displays the number of secure Tx channels. (<i>default = 0</i>)
confidentialityOffset	Used to configure the MacSec port confidentiality offset. (<i>default = 0</i>)
negativeTestingOffset	Used to configure the Tx MacSec port negative testing offset. (<i>default = 0</i>)
negativeTestingMask	Used to configure the Tx MacSec port negative testing mask. (<i>default = '00000000'</i>)

COMMANDS	The macSecTx command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	---

macSecTx setDefault

Sets to IxTclHal default values for all configuration options.

macSecTx select *chasID cardID portID*

Selects the port to set or retrieve data from. By default, it fills in the object with the transmit configuration details.

macSecTx set

Sets the MacSecTx configuraton from IxTclHal to local IxHal object.

macSecTx get

Gets the MacSecTx configuraton from local IxHal to IxTclHal.

macSecTx addChannel

Adds the configured Connectivity Association channel configuration data for this port into the IxHal.

macSecTx getChannel *secureChannelId*

Retrieves the configured Connectivity Association Tx channel configuration data for the specified ID on this port from the IxHal.

macSecTx setChannel *secureChannelId*

Sets the configured Connectivity Association Tx channel configuration data for the specified ID on this port from the IxHal.

macSecTx delChannel *secureChannelId*

Deletes the specified configured Connectivity Association Tx channel configuration data for the specified ID on this port from the IxHal.

macSecTx clearAllChannels

Deletes all the configured Connectivity Association channels for this port from the IxHal.

macSecTx getFirstChannel

Retrieves the first configured Connectivity Association channel configuration data for the selected port from the IxHal.

macSecTx getNextChannel

Retrieves the next configured Connectivity Association channel configuration data for the selected port from the IxHal.

EXAMPLES

See Examples under *macSecTag*.

SEE ALSO

macSecChannel, *macSecRx*, *macSecTag*

NAME - mii

mii - configure the MII parameters for a MII ports

SYNOPSIS

mii *sub-command options*

DESCRIPTION

The **mii** command is used to configure the MII-specific information on old-style IEEE 802.3 devices. New style MII AE devices defined in IEEE 802.3ae are managed by the **miae**, **mmd**, and **mmdRegister** commands.

STANDARD OPTIONS

enableManualAuto
Negotiate true / false If set to *true*, then as the MII register is written to hardware auto negotiation begins. (*default = false*)

miiRegister MII Source register. Defined register values include:

Option	Value	Usage
miiControl	0	(<i>default</i>)
miiStatus	1	
miiPHYId1	2	
miiPHYId2	3	
miiAutoNegAdvertisement	4	
miiAutoNegLinkPartnerAbility	5	
miiAutoNegExpansion	6	
miiRegister7	7	
miiRegister8	8	
miiRegister9	9	
miiRegister10	10	
miiRegister11	11	
miiRegister12	12	
miiRegister13	13	
miiRegister14	14	
miiRegister15	15	
miiMirror	16	
miiInterruptEnable	17	
miiInterruptStatus	18	
miiConfiguration	19	
miiChipStatus	20	
miiRegister21	21	
miiRegister22	22	
miiRegister23	23	
miiRegister24	24	
miiRegister25	25	
miiRegister26	26	

Option	Value	Usage
miiRegister27	27	
miiRegister28	28	
miiRegister29	29	
miiRegister30	30	
miiRegister31	31	

phyAddress

Physical address of the MII register location. If set to `-1`, the default location is used. (*default = -1*)

readWrite

Sets the properties of the selected register. Possible properties include:

Option	Value	Usage
miiDisabled	0	(<i>default</i>)
miiReadOnly	1	
miiReadWrite	2	
miiSyncToCurrentState	3	The register is read and written during operation. In addition, the read values are placed into the editable fields at the same time.

registerValue

Value of the selected register. (*default = 0000*)

COMMANDS

The **mii** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

mii cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **mii** command.

mii config option value

Modify the MII configuration options of the port. If no *option* is specified, returns a list describing all of the available MII options (see STANDARD OPTIONS) for port.

mii get chasID cardID portID [index = \$::mdioInternal]

Gets the current MII configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Any of the three supported PHYs may be selected through the use of the index. The supported PHYs are:

Option	Value	Usage
mdioInternal	0	(<i>default</i>) The internal PHY located on the Ixia card.
mdioExternal1	1	The first defined external PHY.
mdioExternal2	2	The second defined external PHY.

Call this command before calling **mii cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

- The port is being used by another user
- Network error between the client and the chassis

mii selectRegister select

After **mii get** *chasID cardID portID* has completed selects which register to fill the TCL parameters with. Specific errors are:

- No port has previously been selected with the mii.get method
- The port is not an Mii port, or a port with Mii capability

mii set chasID cardID portID [index = \$::mdioInternal]

Sets the MII configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **mii config option value** command. Any of the three supported PHYs may be set through the use of the index. The supported PHYs are:

Option	Value	Usage
mdioInternal	0	(default) The internal PHY located on the Ixia card.
mdioExternal1	1	The first defined external PHY.
mdioExternal2	2	The second defined external PHY.

Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- The port is not an Mii port, or a port with Mii capability

mii setDefault

Sets to IxTclHal default values for all configuration options.

mii write chasID cardID portID

Writes the MII configuration of the port with id *portID* on card *cardID*, chassis *chasID* to the hardware. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Network error between the client and the chassis
- The port is not an Mii port, or a port with Mii capability

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
```

```
        }

    }

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that MII card is in slot 3
set card 3
set portList [list [list $chas $card 1]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Check for missing card
if {[card get $chas $card] != 0} \
{
    ixPuts "Card $card does not exist"
    exit
}

# Get the type of card and check if it's the correct type
set cardType [card cget -type]
if {$cardType != $::card10100Mii} \
{
    ixPuts "Card $card is not a 10/100 MII card"
    exit
}

# Set the options to default values
mii setDefault

# Get the current mii state from the card
mii get $chas $card 1

# Get the value of the control register (0)
mii selectRegister miiControl
set controlReg [mii cget -registerValue]
set msg [format "Register 00 value is %04x" $controlReg]
ixPuts $msg

# Set the mode on register 00 to Read/Write/Sync
mii config -readWrite miiSyncToCurrentState
# With bit 14 (loopback) on
set controlReg [expr $controlReg | 0x0400]
mii config -registerValue $controlReg

# set to ixTclHal
mii set $chas $card 1

# and write to hardware
```

```
set portList [list [list $chas $card 1]]
ixWritePortsToHardware portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

port, miiae, mmd, mmdRegister

NAME - miiae

miiae - configure an MII AE.

SYNOPSIS

miiae *sub-command options*

DESCRIPTION

The **miiae** command is used to configure an MII AE PHY to be associated with a port. **miiae** manages new-style IEEE 802.3ae PHYs. After configuration, **miiae set** should be used to associate it with a port; **port write** or **miiae write** should be used to write the values to the hardware.

STANDARD OPTIONS

phyAddress

The address of the MII AE PHY. (*default* = 31)

COMMANDS

The **miiae** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

miiae addDevice

Adds the device defined through the use of the **mmd** command.

miiae clearAllDevices

Deletes all devices associated with this MII AE PHY.

miiae config *option value*

Modify the configuration options of the PHY. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS).

miiae delDevice *deviceAddress*

Deletes the device whose address is *deviceAddress*.

miiae get *chassisID cardID portID index*

Gets the current MII configuration of the port with id *portID* on card *cardID*, chassis *chassisID*. Any of the three supported PHYs may be selected through the use of the index. The supported PHYs are:

Option	Value	Usage
mdioInternal	0	(<i>default</i>) The internal PHY located on the Ixia card.
mdioExternal1	1	The first defined external PHY.
mdioExternal2	2	The second defined external PHY.

Call this command before calling **mmd** to look at the PHY. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Network error between the client and the chassis

miae getDevice *deviceAddress*

Gets the device whose address is *deviceAddress*. The values associated with the device may be viewed and modified through the use of the **mmd** command.

miae set *chassisID cardID portID index*

Sets the MII configuration of the port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **mmd** command. Any of the three supported PHYs may be set through the use of the index. The supported PHYs are:

Option	Value	Usage
mdioInternal	0	(default) The internal PHY located on the Ixia card.
mdioExternal1	1	The first defined external PHY.
mdioExternal2	2	The second defined external PHY.

Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- The port is not an Mii port, or a port with Mii capability

miae setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts 'Could not connect to $host'
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that a 10GB XAUI card is in slot 35
set card 35
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```

}

# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
set deviceNo 1

# Configure register 1
mmdRegister setDefault
mmdRegister config -address      1
mmdRegister config -name        reg1
mmdRegister config -registerValue 1234
# And add it to the MMD
mmd addRegister

# Configure register 2
mmdRegister config -address      2
mmdRegister config -name        reg2
mmdRegister config -registerValue 3405
mmd addRegister

# Now configure the MMD and add it to the miae
mmd config -address           $deviceNo
mmd config -name              dev1
# Add it to the miae
miae addDevice

miae config -phyAddress        24

# Set and write the miae
if [miae set $chassis $card $port mdioExternal1] {
    ixPuts "Error in miae set"
}
if [miae write $chassis $card $port] {
    ixPuts "Error in miae write"
}
# Now get the object back
if [miae get $chassis $card $port mdioExternal1] {
    ixPuts "Error in miae get"
}
if [miae getDevice $deviceNo] {
    ixPuts "Error in miae getDevice"
}
# Now get the register contents
mmd getRegister 1
set name [mmdRegister cget -name]
set val  [mmdRegister cget -registerValue]
ixPuts "Register 1 ($name) is $val"

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*mii, mmd, mmdRegister*

NAME - mmd

mmd - configure an MII AE PHY.

SYNOPSIS

mmd *sub-command options*

DESCRIPTION

The **mmd** command is used to configure an individual MII AE PHY device. After configuration, **miae addDevice** should be used to add the device to the MII AE. The current contents of the device may be obtained by **miae getDevice**. The value of a device may only be changed by a sequence of **miae getDevice**, **miae delDevice** and **miae addDevice**.

STANDARD OPTIONS

address

The address of the device in the MMD device. (*default* = 0)

name

The name of the device. (*default* = {})

COMMANDS

The **mmd** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

mmd addRegister

Adds the register defined through the use of the **mmdRegister** command to the MMD device.

mmd clearAllRegisters

Deletes all the registers associated with the MMD device.

mmd config option value

Modify the configuration options of the MMD device. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS).

mmd delRegister registerAddress

Deletes the register whose address is *registerAddress*.

mmd getRegister registerAddress

Gets the register whose address is *registerAddress*. The values associated with the register may be viewed and modified through the use of the **mmdRegister** command.

mmd setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples in [miae](#).

SEE ALSO

[mii](#), [miae](#), [mmdRegister](#)

NAME - mmdRegister

mmdRegister - configure an MII AE MMD Register.

SYNOPSIS

`mmdRegister sub-command options`

DESCRIPTION

The **mmdRegister** command is used to configure an individual MII AE MMD register. After configuration, **mmd addRegister** should be used to add the register to the PHY device. The current contents of the register may be obtained by **mmd getRegister**. The value of a register may only be changed by a sequence of **mmd getRegister**, **mmd delRegister** and **mmd addRegister**.

STANDARD OPTIONS

address

The address of the register in the register. (*default* = 0)

name

The name of the register. (*default* = {})

readWrite

Sets the properties of the selected register. Possible properties include:

Option	Value	Usage
miiDisabled	0	
miiReadOnly	1	
miiReadWrite	2	(<i>default</i>)
miiSynchToCurrentState	3	The register is read and written during operation. In addition, the read values are placed into the editable fields at the same time.

registerValue

Value of the selected register. (*default* = 0000)

COMMANDS

The **mmdRegister** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

mmdRegister cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **mmdRegister** command. The value returned for the *registerValue* option is a hex

mmdRegister config option value

Modify the configuration options of the register. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS).

mmdRegister setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples in *miae*.

SEE ALSO

mii, *miae*, *mmd*

NAME - mpls

mpls - configure the MPLS parameters for a port on a card on a chassis

SYNOPSIS	mpls <i>sub-command options</i>
-----------------	---------------------------------

DESCRIPTION	The mpls command is used to configure the MPLS information when building MPLS labeled packets. See draft-ietf-mpls-arch-06.txt “work in progress” for a complete definition of MPLS label fields. Note that <i>stream</i> get must be called before this command’s get sub-command.
--------------------	--

STANDARD OPTIONS

enableAutomaticallySet Label <i>true/false</i>	Sets MPLS to automatically set the label values. (<i>default = true</i>)
---	--

forceBottomOfStack <i>true/false</i>	Automatically sets bottom of the stack bit. (<i>default = true</i>)
---	---

type	Sets the MPLS type. Options include:
-------------	--------------------------------------

Option	Value	Usage
<i>mplsUnicast</i>	0	(<i>default</i>)
<i>mplsMulticast</i>	1	

COMMANDS

The **mpls** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

mpls cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **mpls** command.

mpls config *option value*

Modify the MPLS configuration options of the port. If no *option* is specified, returns a list describing all of the available MPLS options (see STANDARD OPTIONS) for port.

mpls decode *capFrame [chassisID cardID portID]*

Decodes a captured frame in the capture buffer and updates TcHal. **mpls cget** *option* command can be used after decoding to get the option data. Specific errors are:

- No connection to a chassis
- Invalid port number
- The captured frame is not a valid Mpls frame

mpls get *chassisID cardID portID*

Gets the current MPLS configuration of the port with id *portID* on card *cardID*, chassis *chassisID*. Note that *stream* get must be called before this command's get sub-command. Call this command before calling **mpls cget option** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

mpls set *chassisID cardID portID*

Sets the MPLS configuration of the port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **mpls config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

mpls setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set txPort 1
set rxPort 2

# Useful port lists
set portList [list [list $chas $card $txPort] \
                    [list $chas $card $rxPort]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
# Set up Transmit Port

# Nothing special about the ports
port setFactoryDefaults $chassis $card $txPort
port setDefault
port set $chassis $card $txPort
port set $chassis $card $rxPort

# Stream: 10 packets
stream setDefault
stream config -numFrames      10
stream config -dma            stopStream
#stream config -percentPacketRate 100
#stream config -rateMode       usePercentRate

protocol setDefault
protocol config -etherType      ethernetII
protocol config -enableMPLS     true

# Setup up two mpls labels
mpls setDefault
mpls config -type             mplsUnicast

mplsLabel setDefault
mplsLabel config -label        128
mplsLabel config -bottomOfStack false
mplsLabel set 1

mplsLabel config -label        256
mplsLabel config -bottomOfStack true
mplsLabel set 2

mpls set $chassis $card $txPort

stream set $chassis $card $txPort 1

# Commit to hardware
ixWritePortsToHardware portList

# Make sure link is up
after 1000
ixCheckLinkState portList
ixStartPortCapture $chassis $card $rxPort

# Clear stats and transmit MPLS labeled frames
ixClearStats portList
ixStartPortTransmit $chassis $card $txPort

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*stream, protocol, mplsLabel*

NAME - mplsLabel

mplsLabel - configure the MPLS label parameters for a port on a card on a chassis.

SYNOPSIS

`mplsLabel sub-command options`

DESCRIPTION

The **mplsLabel** command is used to configure the MPLS label information when building MPLS labeled packets. See draft-ietf-mpls-arch-06.txt “work in progress” for a complete definition of MPLS label fields.

STANDARD OPTIONS

bottomOfStack
true/false

Enables the bottom of the stack bit. This bit is set to true for the last entry in the label stack (for the bottom of the stack) and false for all other label stack entries. (*default = true*)

experimentalUse

Sets the experimental use bit. (*default=0*)

label

Sets the actual value of the label. Any 20-bit value is valid; predefined options include:

Option	Value	Usage
<i>mplsIPv4ExplicitNULL</i>	0	(<i>default</i>)
<i>mplsRouterAlert</i>	1	
<i>mplsIPv6ExplicitNULL</i>	2	
<i>mplsImplicitNULL</i>	3	
<i>mplsReserved</i>	4	

timeToLive

Sets the time-to-live value. (*default=64*)

COMMANDS

The **mplsLabel** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

mplsLabel cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **mplsLabel** command.

mplsLabel config *option value*

Modify the MPLS label configuration options of the port. If no *option* is specified, returns a list describing all of the available MPLS label options (see STANDARD OPTIONS) for port.

mplsLabel get *labelID*

Gets the current label configuration of the selected *labelID*. Call this command before calling **mplsLabel cget *option*** to get the value of the configuration option. Specific errors are:

A***mplsLabel***

- There are no MPLS labels
- The specified labelID does not exist

mplsLabel set labelID

Sets the label configuration for label labelID reading the configuration option values set by the **mplsLabel config option value** command. Specific errors are:

- The configured parameters are not valid for this port
- Insufficient memory to add a new label

mplsLabel setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under the *mpls* command.

SEE ALSO

stream, protocol, mpls

NAME - networkHeader

networkHeader-configures a network header within the data field of an FC port.

SYNOPSIS

networkHeader *sub-command options*

DESCRIPTION

The **networkHeader** command holds information for a single network header data. NetworkHeader command contains two parts, Destination Address and Source Address, each of which is of six different types and different parameters. The types that represent the Name Identifier Format are as follows:

Destination Address Format Types	Source Address Format Types
<i>IEEE48BitAddressDest</i>	<i>IEEE48BitAddressSrc</i>
<i>IEEEExtendedDest</i>	<i>IEEEExtendedSrc</i>
<i>LocallyAssignedDest</i>	<i>LocallyAssignedSrc</i>
<i>IEEERegisteredDest</i>	<i>IEEERegisteredSrc</i>
<i>IEEERegisteredExtendedDest</i>	<i>IEEERegisteredExtendedSrc</i>
<i>EUI64MappedDest</i>	<i>EUI64MappedSrc</i>

STANDARD OPTIONS

destinationFormat

The format of the name identifier used for the network destination address.

ieee48BitAddressDest

When the Name_Identifier format is IEEE 48-bit Address, the name value contains a 48-bit IEEE Standard 802.1A Universal LAN MAC Address (ULA). The ULA is represented as an ordered string of six bytes numbered from 0 to 5. ULA Bytes 0, 1, and 2 are generated using the IEEE Company_ID.

The name identifier for IEEE 48 Bit Destination Address format is as follows:

Option	Usage
<i>48BitAddressNameIdentifier</i>	<i>The 48 bit address name identifier when the destination address format is IEEE 48 Bit Address.</i>

ieeeExtendedDest

When the Name_Identifier format is IEEE Extended, the name value contains the 48-bit IEEE address preceded by a 12 bit value. The 12 bit value is an extension to the company assigned address portion of the 48-bit address that forms a unique 60-bit value. The 48-bit IEEE address is defined same as for the IEEE 48-bit Address Name_Identifier format.

The name identifier for IEEE Extended Destination Address format is as follows:

Option	Usage
<i>48BitAddressNameIdentifier</i>	<i>The 48 bit address name identifier when the destination address format is IEEE 48 Bit Address.</i>
<i>vendorSpecific</i>	<i>The vendor specific identifier that is mapped with the address format. It is true only when destination address format is IEEE Extended.</i>

locallyAssignedDest

When the Name_Identifier format is locally assigned, the name value field is assigned in a manner determined by the administration of the Fabric in which it is assigned. A locally assigned Name_Identifier is unique within the Fibre Channel interaction space wherein it is assigned.

The name identifier for Locally Assigned Destination Address format is as follows:

Option	Usage
<i>locallyAdministeredValue</i>	<i>The locally administered value that is present only when destination address format is Locally Assigned.</i>

ieeeRegisteredDest

When the Name_Identifier format is IEEE Registered, the name value field contains the 24-bit IEEE Company_ID in canonical form, as specified by IEEE, followed by a 36-bit unique Vendor Specified Identifier (VSID).

The name identifier for IEEE Registered Destination Address format is as follows:

Option	Usage
<i>ieeeCompanyId</i>	<i>The IEEE Company Identifier.</i>
<i>vendorSpecificId</i>	<i>The vendor specific identifier that is mapped with the address format.</i>

ieeeRegisteredExtendedDest

When the Name_Identifier format is IEEE Registered Extended, the name value contains the 24-bit IEEE Company_ID in canonical form, as specified by IEEE, followed by a 36-bit unique vendor specified id (VSID). Name_Identifiers that identify Fibre Channel Nodes or FC_Ports are limited to 64 bits and therefore will not use the IEEE Registered Extended format.

The name identifier for IEEE Registered Extended Destination Address format is as follows:

Option	Usage
<i>ieeeCompanyId</i>	<i>The IEEE Company Identifier.</i>
<i>vendorSpecificId</i>	<i>The vendor specific identifier that is mapped with the address format.</i>
<i>vendorSpecificIdExtension</i>	<i>The vendor specific identifier extension that is present only when destination address format is IEEE Registered Extended.</i>

eui64MappedDest

When the Name_Identifier format is EUI64 Mapped, The NAA field contains either 0Ch, 0Dh, 0Eh, or 0Fh. The name value field contains a modified 22-bit IEEE Company_ID, followed by a 40-bit unique VSID.

The name identifier for EUI64 Mapped Destination Address format is as follows:

Option	Usage
<i>ieeeCompanyId</i>	<i>The IEEE Company Identifier.</i>
<i>vendorSpecificId</i>	<i>The vendor specific identifier that is mapped with the address format.</i>

sourceFormat

The format of the name identifier used for the network source address.

ieee48BitAddressSrc

When the Name_Identifier format is IEEE 48-bit Address, the name value contains a 48-bit IEEE Standard 802.1A Universal LAN MAC Address (ULA). The ULA is represented as an ordered string of six bytes numbered from 0 to 5. ULA Bytes 0, 1, and 2 are generated using the IEEE Company_ID.

The name identifier for IEEE 48 Bit Source Address format is as follows:

Option	Usage
<i>48BitAddressNameIdentifier</i>	<i>The 48 bit address name identifier when the source address format is IEEE 48 Bit Address.</i>

ieeeExtendedSrc

When the Name_Identifier format is IEEE Extended, the name value contains the 48-bit IEEE address preceded by a 12 bit value. The 12 bit value is an extension to the company assigned address portion of the 48-bit address that forms a unique 60-bit value. The 48-bit IEEE address is defined same as for the IEEE 48-bit Address Name_Identifier format.

The name identifier for IEEE Extended Source Address format is as follows:

Option	Usage
<i>48BitAddressNameIdentifier</i>	<i>The 48 bit address name identifier when the source address format is IEEE 48 Bit Address.</i>
<i>vendorSpecific</i>	<i>The vendor specific identifier that is mapped with the address format. It is true only when source address format is IEEE Extended.</i>

locallyAssignedSrc

When the Name_Identifier format is locally assigned, the name value field is assigned in a manner determined by the administration of the Fabric in which it is assigned. A locally assigned Name_Identifier is unique within the Fibre Channel interaction space wherein it is assigned.

The name identifier for Locally Assigned Source Address format is as follows:

Option	Usage
<i>locallyAdministeredValue</i>	<i>The locally administered value that is present only when source address format is Locally Assigned.</i>

ieeeRegisteredSrc

When the Name_Identifier format is IEEE Registered, the name value field contains the 24-bit IEEE Company_ID in canonical form, as specified by IEEE, followed by a 36-bit unique Vendor Specified Identifier (VSID).

The name identifier for IEEE Registered Source Address format is as follows:

Option	Usage
<i>ieeeCompanyId</i>	<i>The IEEE Company Identifier.</i>
<i>vendorSpecificId</i>	<i>The vendor specific identifier that is mapped with the address format.</i>

ieeeRegisteredExtendedSrc

When the Name_Identifier format is IEEE Registered Extended, the name value contains the 24-bit IEEE Company_ID in canonical form, as specified by IEEE, followed by a 36-bit unique vendor specified id (VSID). Name_Identifiers that identify Fibre Channel Nodes or FC_Ports are limited to 64 bits and therefore will not use the IEEE Registered Extended format.

The name identifier for IEEE Registered Extended Source Address format is as follows:

Option	Usage
<i>ieeeCompanyId</i>	<i>The IEEE Company Identifier.</i>
<i>vendorSpecificId</i>	<i>The vendor specific identifier that is mapped with the address format.</i>
<i>vendorSpecificId Extension</i>	<i>The vendor specific identifier extension that is present only when source address format is IEEE Registered Extended.</i>

eui64MappedSrc

When the Name_Identifier format is EUI64 Mapped, The NAA field contains either 0Ch, 0Dh, 0Eh, or 0Fh. The name value field contains a modified 22-bit IEEE Company_ID, followed by a 40-bit unique VSID.

The name identifier for EUI64 Mapped Source Address format is as follows:

Option	Usage
<i>ieeeCompanyId</i>	<i>The IEEE Company Identifier.</i>
<i>vendorSpecificId</i>	<i>The vendor specific identifier that is mapped with the address format.</i>

COMMANDS

The **networkHeader** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

networkHeader *setDefault* *option*

Sets to default values for all configuration options.

networkHeader *setDestination* *option*

Sets the destination address format of the network header.

networkHeader *setSource*

Sets the source address format of the network header.

networkHeader *getDestination*

Gets the destination address format of the network header.

networkHeader *getSource*

Gets the source address format of the network header.

networkHeader *decodeDestination*

Decodes the destination address format configuration options for the network header.

networkHeader *decodeSource*

Decodes the source address format configuration options for the network header.

EXAMPLES

See examples under the *fibreChannel* command.

SEE ALSO

fibreChannel

NAME - npiProperties

npiProperties - configure unconnected NPIV interface.

SYNOPSIS	npiProperties <i>sub-command options</i>
-----------------	--

DESCRIPTION	The npiProperties command is used to configure an unconnected NPIV interface. (NPIV means N_Port_ID Virtualization).
--------------------	---

STANDARD OPTIONS

destinationId	Destination Identifier (<i>default = 01.b6.69</i>)
bufferToBufferRxSize	Maximum buffer-to-buffer Receive_Data_Field specified by the Fabric (<i>default = 2112</i>)
enableAutoPlogi	Automatically enables PLOGI to all the ports that are advertised by the fabric, or to PLOGI to a subset of the variable ports that belong to a specified domain. (<i>default = false</i>)
enableNs <i>true/false</i>	Enables registration to Name Server (<i>default = false</i>)
enableNSQuery	If true, enables Name Server Query parameters for this FCoE server.
enablePlogi <i>true/false</i>	Enables Port login to specified Destination ID (<i>default = false</i>)
enablePRLI	If true, enables Process Login parameters. The PRLI request is used to establish the operating environment between a group of related processes at the originating Nx_Port and a group of related processes at the responding Nx_Port. If true, this option causes the state machine to attempt a process login.
enableSCR <i>true/false</i>	If set to true, the ENode registers for any changes with the Fabric by sending a State Change Registration packet. (<i>default = false</i>)
enableVnPortKeep Alives	If true, VN port sends periodic keep alives.
scrOption	The State Change Registration (SCR) function options. The options are as follows:

Option	Usage
<i>fabricDetectedRegistration</i>	Register to receive all RSCN Requests issued by the Fabric Controller for events detected by the Fabric.
<i>nxPortDetectedRegistration</i>	Register to receive all RSCN Requests issued for events detected by the affected Nx_Port.

Option	Usage
<i>fullRegistration</i>	Register to receive all RSCN Requests issued. The RSCN Request returns all affected N_Port_IDs.

sourceNodeWWN	Source node Worldwide Name - a Name_identifier that is worldwide unique, represented by a 64-bit value. (<i>default = '00 ... 00'</i>)
sourcePortWWN	Source port Worldwide Name - a Name_identifier that is worldwide unique, represented by a 64-bit value. (<i>default = '00 ... 00'</i>)

COMMANDS	The npiProperties command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

npiProperties cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **npiProperties** command.

npiProperties config *option value*

Modify the NPIV properties configuration options of the port. If no *option* is specified, returns a list describing all of the available NPIV properties options (see STANDARD OPTIONS) for port.

npiProperties setDefault

Sets to IxTclHal default values for all configuration options.

npiProperties addPlogi

Adds a PLOGI to npiProperties. The values are available in the [*fcoePlogi*](#) command.

npiProperties delPlogi *plogiIndex*

Deletes the PLOGI associated with this NPIV property set at the specified index. The index of the first entry is 1. The values are available in the [*fcoePlogi*](#) command. Specific errors are:

- The indexed entry does not exist in the list.
- Invalid index.

npiProperties getPlogi *plogiIndex*

Retrieves the PLOGI associated with this NPIV property set at the specified index. The index of the first entry is 1. The values are available in the [*fcoePlogi*](#) command. Specific errors are:

- The indexed entry does not exist in the list.

npiProperties getFirstPlogi

Retrieves the first PLOGI associated with this NPIV property set. The values are available in the [*fcoePlogi*](#) command. Specific errors are:

- There are no entries in the list.

npiProperties `getNextPlogi`

Retrieves the next PLOGI associated with this NPIV property set. The values are available in the *fcoePlogi* command. Specific errors are:

- There are no more entries in the list.

npiProperties `removeAllPlogis`

Deletes all of the PLOGIs associated with this NPIV property set.

EXAMPLES

See example under *fcoe*.

SEE ALSO

fcoe, fcoeDiscoveredInfo, fcoeProperties, fibreChannel, fcoePlogi

NAME - oamEventNotification

oamEventNotification - the OAM PDU type *Event Notification*.

SYNOPSIS

oamEventNotification *sub-command options*

DESCRIPTION

The **oamEventNotification** command implements the OAM PDU type *Event Notification*.

The TLVs connected to this command include: Errored Symbol Period, Errored Frame, Errored Frame Period, Errored Frame Seconds Summary, and Organization Specific.

STANDARD OPTIONS

currentTlvType

Read only. (default = oamEventNotificationEndOfTlv)

Predefined options include:

Option	Value	Usage
oamEventNotificationEndOfTlv	0x00	(default) End of TLV Marker
oamEventNotificationSymbol	0x01	Errored Symbol Period Event
oamEventNotificationFrame	0x02	Errored Frame Event
oamEventNotificationFramePeriod	0x03	Errored Frame Period Event
oamEventNotificationSummary	0x04	Errored Frame Seconds Summary Event
oamEventNotificationOrgSpecific	0xFE	Organization Specific Event

sequenceNumber

The OAM client increments the Sequence Number for each unique Event Notification OAMPDU formed by the OAM client. A particular Event Notification OAMPDU may be sent multiple times with the same sequence number. Upon receiving an Event Notification OAMPDU, the OAM client compares the Sequence Number with the last received Sequence Number. If equal, the current event is a duplicate and is ignored by the OAM client. (*default = 0*)

COMMANDS

The **oamEventNotification** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamEventNotification **addTlv *tlvType***

Adds a TLV to OAM Event Notification PDU with TLV type (see *currentTlvType*, above).

oamEventNotification **delTlv *tlvIndex***

Deletes a TLV from OAM Event Notification with specific Index.

oamEventNotification *setTlv tlvIndex*

Sets the configuration of the TLV with the specified Index.

oamEventNotification *getTlv tlvIndex*

Gets the configuration of the TLV with the specified Index.

oamEventNotification *getFirstTlv*

Gets the first TLV from the list of OAM Event Notification PDUs.

oamEventNotification *getNextTlv*

Gets the next TLV from the list.

oamEventNotification *clearAllTlvs*

Clears all TLVs for the Event Notification PDU.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, *oamSymbolPeriodTlv*, *oamFrameTlv*, *oamFramePeriodTlv*,
oamSummaryTlv, *oamOrganizationSpecificTlv*, *oamEventOrgTlv*

NAME - oamEventOrgTlv

oamEventOrgTlv - implements one type of OAM Event Notification PDU.

SYNOPSIS

oamEventOrgTlv *sub-command options*

DESCRIPTION

The **oamEventOrgTlv** command implements one type of OAM Event Notification PDU. The Organization Specific Event TLV is used for vendor extensions. The 32-bit vendor specific information is not defined and is used to encode the model or version of the platform.

STANDARD OPTIONS

type	<i>Read only</i> . Set to 254 (0xFE) to indicate Organization Specific Event.
length	<i>Read only</i> . Set to 16 (0x10). The length (in octets) of this TLV-tuple.
oui	Organization unique identifier. (<i>default</i> = '00 00 00')
organizationSpecific Value	The value of the Organization Specific Information TLV (typically, the model or version of the platform).

COMMANDS

The **oamEventOrgTlv** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamEventOrgTlv setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, oamEventNotification, oamSymbolPeriodTlv, oamFrameTlv, oamFramePeriodTlv, oamSummaryTlv, oamOrganizationSpecificTlv

NAME - oamFrameTlv

oamFrameTlv - implements one type of OAM Event Notification PDU.

SYNOPSIS	<i>oamFrameTlv sub-command options</i>
DESCRIPTION	<p>The oamFrameTlv command implements one type of OAM Event Notification PDU.</p> <p>The <i>Errored Frame Event TLV</i> counts the number of errored frames detected during the specified period. The period is specified by a time interval. This event is generated if the errored frame count is equal to or greater than the specified threshold for that period. This event is generated at the end of the event window rather than when the threshold is crossed.</p>
STANDARD OPTIONS	
length	<i>Read only.</i> This one-octet field set to 26 (0x1A). Indicates the length (in octets) of this TLV_tuple.
timestamp	This two-octet field indicates the time reference when the event was generated, in terms of 100 ms intervals, encoded as a 16-bit unsigned integer. (<i>default = 0</i>)
frames	This four-octet field indicates the number of detected errored frames in the period, encoded as a 32-bit unsigned integer. (<i>default = 0</i>)
window	<p>This two-octet field indicates the duration of the period, in terms of 100 ms intervals, encoded as a 16-bit unsigned integer. (<i>default = 0</i>)</p> <p>Lower bound: one second. Upper bound: one minute.</p>
threshold	<p>This four-octet field indicates the number of detected errored frames in the period is required to be equal to or greater than in order for the event to be generated, encoded as a 32-bit unsigned integer. (<i>default = 0</i>)</p> <p>Lower bound: zero symbol errors. Upper bound: unspecified.</p>
eventRunningTotal	This four-octet field indicates the number of Errorred Frame Event TLVs that have been generated since the OAM sublayer was reset, encoded as a 32-bit unsigned integer. (<i>default = 0</i>)
errorRunningTotal	This eight-octet field indicates the sum of errored frames that have been detected since the OAM sublayer was reset. (<i>default = 0</i>)
COMMANDS	The oamFrameTlv command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamFrameTlv setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, *oamEventNotification*, *oamSymbolPeriodTlv*, *oamEventOrgTlv*,
oamFramePeriodTlv, *oamSummaryTlv*, *oamOrganizationSpecificTlv*

NAME - oamFramePeriodTlv

oamFramePeriodTlv - implements one type of OAM Event Notification PDU.

SYNOPSIS	<code>oamFramePeriodTlv sub-command options</code>
DESCRIPTION	<p>The oamFramePeriodTlv command implements one type of OAM Event Notification PDU.</p> <p>The <i>Errored Frame Period Event TLV</i> counts the number of errored frames detected during the specified period. The period is specified by a number of received frames. This event is generated if the errored frame count is greater than or equal to the specified threshold for that period (for example, if the errored frame count is greater than or equal to 10 for the last 1,000,000 frames received). This event is generated at the end of the event window rather than when the threshold is crossed.</p>
STANDARD OPTIONS	
length	<i>Read only.</i> This one-octet field set to 286 (0x1C). Indicates the length (in octets) of this TLV_tuple.
timestamp	This two-octet field indicates the time reference when the event was generated, in terms of 100 ms intervals, encoded as a 16-bit unsigned integer. (<i>default = 0</i>)
window	<p>This four-octet field indicates the duration of the period, in terms of frames, encoded as a 32-bit unsigned integer. (<i>default = 0</i>)</p> <p>Lower bound: the number of minFrameSize frames that can be received in 100 ms on the underlying physical layer.</p> <p>Upper bound: the number of minFrameSize frames that can be received in one minute on the underlying physical layer.</p>
frames	This four-octet field indicates the number of detected errored frames in the period, encoded as a 32-bit unsigned integer. (<i>default = 0</i>)
threshold	<p>This four-octet field indicates the number of detected errored frames in the period is required to be equal to or greater than in order for the event to be generated, encoded as a 32-bit unsigned integer. (<i>default = 0</i>)</p> <p>Lower bound: zero symbol errors.</p> <p>Upper bound: unspecified.</p>
errorRunningTotal	This eight-octet field indicates the sum of errored frames that have been detected since the OAM sublayer was reset. (<i>default = 0</i>)
eventRunningTotal	This four-octet field indicates the number of Errorred Frame Event TLVs that have been generated since the OAM sublayer was reset, encoded as a 32-bit unsigned integer. (<i>default = 0</i>)

COMMANDS

The **oamFramePeriodTlv** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamFramePeriodTlv setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, *oamEventNotification*, *oamSymbolPeriodTlv*, *oamEventOrgTlv*,
oamFrameTlv, *oamSummaryTlv*, *oamOrganizationSpecificTlv*

NAME - oamHeader

oamHeader - configure the OAM header for streams.

SYNOPSIS	<i>oamHeader sub-command options</i>
-----------------	--------------------------------------

DESCRIPTION	The oamHeader command is used to for the stream configuration for the OAM header. Port configuration for OAM is implemented by oamPort . The remaining OAM objects are tightly coupled to this object. This object lives on the protocolStack, consistent with all other stream protocol configuration objects.
--------------------	---

STANDARD OPTIONS

type	<i>Read only</i> . Set to 34825 = Slow Protocols.
subType	<i>Read only</i> . Set to 3 (0x03 = OAM).
flags	Byte, or'd value with enums. 2-byte flag field contains the discovery status of local and remote OAM entities, as well as fault indications. (<i>default</i> = 0) Example: <code>oamHeader config -flags [expr \$::oamFlagCriticalEvent \$::oamFlagLocalEvaluating]</code>
	Predefined options include:

Option	Value	Usage
<i>oamFlagNone</i>	0x0000	(<i>default</i>)
<i>oamFlagLinkFault</i>	0x0001	Link Fault
<i>oamFlagDyingGasp</i>	0x0002	Dying Gasp
<i>oamFlagCriticalEvent</i>	0x0004	Critical Event
<i>oamFlagLocalEvaluating</i>	0x0008	Local Evaluating
<i>oamFlagLocalStable</i>	0x0010	Local Stable
<i>oamFlagRemoteEvaluating</i>	0x0020	Remote Evaluating
<i>oamFlagRemoteStable</i>	0x0040	Remote Stable

code	PDU types. (<i>default</i> = <i>oamCodeInformation</i>)
	Predefined options include:

Option	Value	Usage
<i>oamCodeInformation</i>	0x00	Information
<i>oamCodeEventNotification</i>	0x01	Event Notification
<i>oamCodeVariableRequest</i>	0x02	Variable Request
<i>oamCodeVariableResponse</i>	0x03	Variable Response
<i>oamCodeLoopbackControl</i>	0x04	Loopback Control
<i>oamCodeOrgSpecific</i>	0xFE	Organization Specific

COMMANDS

The **oamHeader** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamHeader setDefault

Sets to IxTclHal default values for all configuration options.

oamHeader set chasID cardID portID

Sets the OAM header and family of OAM objects into IxHal.

oamHeader get chasID cardID portID

Retrieves the OAM header and family of OAM objects from IxHal.

oamHeader decode capFrame chasID cardID portID

Decodes the OAM stream.

EXAMPLES

```

package req IxTclHal
set hostname astro
ixConnectToChassis $hostName
set retCode "PASS"

if {[ixConnectToChassis $hostName] } {
    errorMsg "Error connecting to $hostName"
    set retCode "FAIL"
}
set chassisId [chassis cget -id]
set portList [list]

for { set cardId 1 } {$cardId <= [chassis cget -maxCardCount] }
{incr cardId} {
    if {[card get $chassisId $cardId] == $::TCL_OK} {
        set portId 1
        if {[port isValidFeature $chassisId $cardId $portId
portFeatureMACSec]} {
            port setModeDefaults $chassisId $cardId $portId
            lappend portList [list $chassisId $cardId $portId]
        }
    }
}
if {[llength $portList] == 0} {
    errorMsg "No ports in port list that support MACSec"
    set retCode "FAIL"
    return $retCode
}

foreach port $portList {
    scan $port "%d %d %d" chassis card port

    set streamId1

    oamPort setDefault
    oamPort config -enable
    oamPort config -macAddress
    AD"
    oamPort config -enableLoopback
    oamPort config -enableLinkEvents
    true
    "00 00 AB BA DE
    true
    true
}

```

```

oamPort config -maxOamPduSize          1518
oamPort config -oui                  "00 00 00"
oamPort config -vendorSpecificInformation      "00 00 00 00"
oamPort config -idleTimer            5
oamPort config -enableOptionalTlv        true
oamPort config -optionalTlvType        254
oamPort config -optionalTlvValue       "11 11 11 11 11"
oamPort set   $chassis $card $port
lappend portList           [list $chassis $card $port]
ixWritePortsToHardware    portList
ixCheckLinkState         portList

# Stream 1
stream setDefault
stream config -name      "OamStream"
stream config -enable     true
stream config -framesize  200

protocol setDefault
protocol config -enableOAM  true

oamHeader setDefault
oamHeader config -flags [expr
$::oamFlagCriticalEvent|$::oamFlagLocalEvaluating|$::oamFlagLocal
Stable|$::oamFlagRemoteEvaluation]
oamHeader config -code  oamCodeLoopbackControl

oamInformation           clearAllTlvs

oamLocalInformationTlv setDefault
oamLocalInformationTlv config -revision 0
oamLocalInformationTlv config -parserAction
oamParserActionForward
oamLocalInformationTlv config -multiplexerAction
oamMultiplexerActionForward
oamLocalInformationTlv config -enableOamPassiveMode false
oamLocalInformationTlv config -enableUnidirectional false
oamLocalInformationTlv config -enableLinkEvents false
oamLocalInformationTlv config -enableRemoteLoopback false
oamLocalInformationTlv config -enableVariableRetrieval false
oamLocalInformationTlv config -maxPduSize 0
oamLocalInformationTlv config -oui "00 00 00"
oamLocalInformationTlv config -vendorSpecificInformation "00 00 00
00"

if {[oamInformation addTlv oamInformationLocalInfo]} {
    errorMsg "Error adding oamEventNotification
oamInformationLocalInfo TLV "
}

oamRemoteInformationTlv setDefault
oamRemoteInformationTlv config -revision 0
oamRemoteInformationTlv config -parserAction
oamParserActionForward
oamRemoteInformationTlv config -multiplexerAction
oamMultiplexerActionForward
oamRemoteInformationTlv config -enableOamPassiveMode false
oamRemoteInformationTlv config -enableUnidirectional false
oamRemoteInformationTlv config -enableLinkEvents false
oamRemoteInformationTlv config -enableRemoteLoopback false

```

```

oamRemoteInformationTlv config -enableVariableRetrieval false
oamRemoteInformationTlv config -maxPduSize 0
oamRemoteInformationTlv config -oui "00 00 00"
oamRemoteInformationTlv config -vendorSpecificInformation "00 00
00 00"

if {[oamInformation addTlv oamInformationRemoteInfo]} {
    errorMsg "Error adding oamEventNotification
oamInformationRemoteInfo TLV "
}

oamEventNotification      setDefault
oamEventNotification      clearAllTlvs
oamEventNotification      config -sequenceNumber 0

oamSymbolPeriodTlv setDefault
oamSymbolPeriodTlv config -symbols          10
oamSymbolPeriodTlv config -timestamp        10
oamSymbolPeriodTlv config -window           10
oamSymbolPeriodTlv config -threshold         10
oamSymbolPeriodTlv config -errorRunningTotal 10
oamSymbolPeriodTlv config -eventRunningTotal 10

if {[oamEventNotification addTlv oamEventNotificationSymbol]} {
    errorMsg "Error adding oamEventNotification
oamEventNotificationSymbol TLV "
}

oamSummaryTlv setDefault
oamSummaryTlv config -frameSeconds          10
oamSummaryTlv config -timestamp            10
oamSummaryTlv config -window              110
oamSummaryTlv config -threshold           110
oamSummaryTlv config -errorRunningTotal   110
oamSummaryTlv config -eventRunningTotal    110

if {[oamEventNotification addTlv oamEventNotificationSummary]} {
    errorMsg "Error adding oamEventNotification
oamEventNotificationSummary TLV "
}

oamEventOrgTlv setDefault
oamEventOrgTlv config -oui                  "00 00 00"
oamEventOrgTlv config -length               16
oamEventOrgTlv config -organizationSpecificValue "00 00 00
00 00 00 00 00 00 00 00"

if {[oamEventNotification addTlv
oamEventNotificationOrgSpecific]} {
    errorMsg "Error adding oamEventNotification
oamEventNotificationOrgSpecific TLV "
}

oamVariableRequest      clearAllTlvs
oamVariableResponse      clearAllTlvs

oamLoopbackControl      setDefault
oamLoopbackControl      config -enableLoopback true

oamOrganizationSpecific setDefault
oamOrganizationSpecific config -oui "00 00 00"

```

```

oamOrganizationSpecific      config      -
organizationSpecificValue    "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

if {[oamHeader set $chassis $card $port]} {
    errorMsg "Error setting oam header on $chassis $card
$port"
}

if {[stream set $chassis $card $port $streamId]} {
    errorMsg "Error setting oam header on $chassis $card
$port"
}

ixWriteConfigToHardware      portList      -noProtocolServer

```

SEE ALSO

oamInformation, oamEventNotification, oamVariableRequest, oamVariableResponse, oamLoopbackControl, oamOrganizationSpecific

NAME - oamInformation

oamInformation - the OAM PDU type *Information*.

SYNOPSIS

oamInformation *sub-command options*

DESCRIPTION

The **oamInformation** command implements the OAM PDU type *Information*.

The TLVs connected to this command include: Local Information, Remote Information, and *oamOrganizationSpecificTlv*.

STANDARD OPTIONS

currentTlvType

Read only. (default = oamInformationEndOfTlv)

Predefined options include:

Option	Value	Usage
oamInformationEndOfTlv	0x00	(default) End of TLV Marker
oamInformationLocalInfo	0x01	Local Information TLV
oamInformationRemoteInfo	0x02	Remote Information TLV
oamInformationOrgInfo	0xFE	Organization Specific TLV

COMMANDS

The **oamInformation** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamInformation **addTlv *tlvType***

Adds a TLV to OAM Information PDU with TLV type (listed above in *currentTlvType*).

oamInformation **delTlv *tlvIndex***

Deletes a TLV from OAM Information with specific Index.

oamInformation **setTlv *tlvIndex***

Sets the configuration of the TLV with the specified Index.

oamInformation **getTlv *tlvIndex***

Gets the configuration of the TLV with the specified Index.

oamInformation **getFirstTlv**

Gets the first TLV from the list of OAM Information PDUs.

oamInformation **getNextTlv**

Gets the next TLV from the list.

oamInformation **clearAllTlvs**

Clears all TLVs for the Information PDU.

A***oamInformation*****EXAMPLES**

See example code under [*oamHeader*](#).

SEE ALSO

[*oamHeader*](#), [*oamLocalInformationTlv*](#), [*oamRemoteInformationTlv*](#),
[*oamOrganizationSpecificTlv*](#)

NAME - oamLocalInformationTlv

oamLocalInformationTlv - implements one type of OAM Information TLV.

SYNOPSIS

oamLocalInformationTlv *sub-command options*

DESCRIPTION

The **oamLocalInformationTlv** command implements one type of OAM Information PDU. Local and remote information is used in the discovery process.

STANDARD OPTIONS

length

Read only. Set to 0. Indicates the length (in octets) of this TLV_tuple.

enableLinkEvents
true/false

Enable to interpret link events. (*default = false*)

enableOamPassive
Mode *true/false*

DTE configured in active (*true*) or passive mode. (*default = false*)

enableRemote
Loopback *true/false*

Enable OAM remote loopback mode. (*default = false*)

enableUnidirectional
true/false

OAM provides an OAM PDU-based mechanism to notify the remote DTE when one direction of a link is non-operational and therefore data transmission is disabled. The ability to operate a link in a unidirectional mode for diagnostic purposes supports failure detection and notification. (*default = false*)

enableVariable
Retrieval *true/false*

Enable variable retrieval (see *oamVariableRequest*). (*default = false*)

maxPduSize

11-bit field which represents the largest OAM PDU, in octets, supported by the DTE. This value is compared to the remote's Maximum PDU Size and the smaller of the two is used. (*default = 0*)

multiplexerAction

Multiplexer function is responsible for passing frames received from the superior sublayer (for example, MAC client sublayer), OAMPDUs from the Control function and loopback frames from the Parser, to the subordinate sublayer (for example, MAC sublayer). (*default = oamMultiplexerActionForward*)

Option	Value	Usage
oamMultiplexerActionForward	0x00	(default) sends on the request over the wire
oamMultiplexerActionDiscard	0x01	discards the request

parserAction

Parser distinguishes among OAMPDUs, MAC client frames and loopback frames and passes each to the appropriate entity (Control, superior sublayer and Multiplexer, respectively). (*default = oamParserActionForward*)

Option	Value	Usage
oamParserActionForward	0x00	(default) lower layer forwards request to upper layer
oamParserActionLoopback	0x01	lower layer sends back request
oamParserActionDiscard	0x02	lower layer discards request
oamParserActionInvalid	0x03	parser action is invalid

revision

The current revision of the Information TLV. The value of this field shall start at zero and be incremented each time something in the Information TLV changes. (*default = 0*)

oamVersion

Read only. Set to 1.

oui

Organization unique identifier. 3 hex bytes. (*default = '00 00 00'*)

vendorSpecific Information

An unspecified list of hex bytes. May be used to differentiate a vendor's product models/versions. (*default = '00 00 00 00'*)

COMMANDS

The **oamLocalInformationTlv** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamLocalInformationTlv setDefault

Sets to IxTclHal default values for all configuration options for this command.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, oamInformation, oamRemoteInformationTlv, oamOrganizationSpecificTlv

NAME - oamLoopbackControl

oamLoopbackControl - the OAM PDU type *Loopback Control*.

SYNOPSIS oamLoopbackControl *sub-command options*

DESCRIPTION The **oamLoopbackControl** command implements the OAM PDU type *Loopback Control*.

STANDARD OPTIONS

enableLoopback Enable/disable Loopback control. (*default = false*)
true/false

COMMANDS The **oamLoopbackControl** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamLoopbackControl setDefault

Sets to IxTclHal default values for all configuration options for this command.

EXAMPLES See example code under *oamHeader*.

SEE ALSO *oamHeader*

NAME - **oamOrganizationSpecific**

oamOrganizationSpecific - the OAM PDU type *Organization Specific*.

SYNOPSIS *oamOrganizationSpecific sub-command options*

DESCRIPTION The **oamOrganizationSpecific** command implements the OAM PDU type *Organization Specific*, which is used for vendor extensions.

STANDARD OPTIONS

oui Organization unique identifier. 3 hex bytes. (*default = '00 00 00'*)

organizationSpecific Value 39-byte hex value of all zeroes ('00 ... 00')

COMMANDS The **oamOrganizationSpecific** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamOrganizationSpecific setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES See example code under *oamHeader*.

SEE ALSO *oamHeader, oamLocalInformationTlv, oamRemoteInformationTlv, oamOrganizationSpecificTlv*

NAME - oamOrganizationSpecificTlv

oamOrganizationSpecificTlv - implements one type of OAM Information PDU.

SYNOPSIS

oamOrganizationSpecificTlv *sub-command options*

DESCRIPTION

The **oamOrganizationSpecificTlv** command implements one type of OAM Information PDU. It is used for vendor extensions. The 32-bit vendor specific information is not defined and is used to encode the model or version of the platform.

STANDARD OPTIONS

type	<i>Read only.</i> Set to 254 (0xFE) to indicate Organization Specific Information.
length	<i>Read only.</i> The length (in octets) of an Organization Specific Information TLV is set to 16.
oui	Organization unique identifier. 3 hex bytes. (<i>default = '00 00 00'</i>)
organizationSpecificValue	11-bytes hex list. The value of the Organization Specific Information TLV. (<i>default is all zeroes</i>)

COMMANDS

The **oamOrganizationSpecificTlv** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamOrganizationSpecificTlv **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, oamInformation, oamRemoteInformationTlv, oamLocalInformationTlv

NAME - oamPort

oamPort - configure the OAM port properties.

SYNOPSIS

oamPort sub-command options

DESCRIPTION

The **oamPort** command is used to configure the OAM port properties. Stream configuration for OAM is implemented by *oamHeader*.

STANDARD OPTIONS

enable <i>true/false</i>	Enables/disables the OAM on the port. (<i>default = false</i>)															
macAddress	Allows configuration of the OAM Mac address on the port. (<i>default = '00 00 00 00 00 00'</i>)															
enableLoopback <i>true/false</i>	Enables/disables the OAM loopback capabilities. (<i>default = false</i>)															
enableLinkEvents <i>true/false</i>	Enables/disables the OAM link events capabilities. (<i>default = false</i>)															
maxOamPduSize	Allows configuration of the maximum OAM PDU size. (<i>default = 1518</i>)															
oui	Allows configuration of the organization unique identifier. (<i>default = '00 00 00'</i>)															
vendorSpecificInformation	Allows configuration of the vendor specific information. (<i>default = '00 00 00 00'</i>)															
idleTimer	Allows configuration of the idle timer (in seconds). (<i>default = 5</i>)															
enableOptionalTlv	Enable/disable optional TLV. (<i>default = false</i>)															
optionalTlvType	Applies only when <i>enableOptionalTlv</i> is set to <i>true</i> . (<i>default = 254</i> , which is <i>oamInformationOrgInfo</i>):															
	<table border="1"> <thead> <tr> <th>Option</th> <th>Value</th> <th>Usage</th> </tr> </thead> <tbody> <tr> <td><i>oamInformationEndOfTlv</i></td> <td>0x00</td> <td>(default) End of TLV Marker</td> </tr> <tr> <td><i>oamInformationLocalInfo</i></td> <td>0x01</td> <td>Local Information TLV</td> </tr> <tr> <td><i>oamInformationRemoteInfo</i></td> <td>0x02</td> <td>Remote Information TLV</td> </tr> <tr> <td><i>oamInformationOrgInfo</i></td> <td>0xFE</td> <td>Organization Specific TLV</td> </tr> </tbody> </table>	Option	Value	Usage	<i>oamInformationEndOfTlv</i>	0x00	(default) End of TLV Marker	<i>oamInformationLocalInfo</i>	0x01	Local Information TLV	<i>oamInformationRemoteInfo</i>	0x02	Remote Information TLV	<i>oamInformationOrgInfo</i>	0xFE	Organization Specific TLV
Option	Value	Usage														
<i>oamInformationEndOfTlv</i>	0x00	(default) End of TLV Marker														
<i>oamInformationLocalInfo</i>	0x01	Local Information TLV														
<i>oamInformationRemoteInfo</i>	0x02	Remote Information TLV														
<i>oamInformationOrgInfo</i>	0xFE	Organization Specific TLV														
optionalTlvValue	Applies only when <i>enableOptionalTlv</i> is set to <i>true</i> .															

COMMANDS

The **oamPort** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamPort setDefault

Sets to IxTclHal default values for all configuration options.

oamPort set chasID cardID portID

Sets the OAM configuration into the port.

oamPort get chasID cardID portID

Retrieves the configured OAM from the port.

EXAMPLES

```
package req IxTclHal

set hostname astro
ixConnectToChassis $hostname

if {[ixConnectToChassis $hostName] } {
    errorMsg "Error connecting to $hostName"
    set retCode "FAIL"
}
set chassisId [chassis cget -id]
set portList [list]

for { set cardId 1 } {$cardId <= [chassis cget -maxCardCount] }
{incr cardId} {
    if {[card get $chassisId $cardId] == $::TCL_OK} {
        set portId 1
        if {[port isValidFeature $chassisId $cardId $portId
portFeatureOam] } {
            port setModeDefaults $chassisId $cardId $portId
            lappend portList [list $chassisId $cardId $portId]
        }
    }
}
if {[llength $portList] == 0} {
    errorMsg "No ports in port list that support portFeatureOam"
    set retCode "FAIL"
    return $retCode
}

foreach port $portList {
    scan $port "%d %d %d" chassisId cardId portId

        oamPort config -enable $::true
        oamPort config -macAddress"01 02 03 aa bb cc"
        oamPort config -enableLoopback$::true
        if {[oamPort set $chassisId $cardId $portId]} {
            errorMsg "Error setting oamPort on $chassisId
$cardId $portId"
        }
}

ixWriteConfigToHardware portList

foreach port $portList {
    scan $port "%d %d %d" chassisId cardId portId

        if {[oamStatus get $chassisId $cardId $portId]} {
            errorMsg "Error getting oamStatus on $chassisId
$cardId $portId"
        }
        if {[oamStatus getLocalStatus]} {
```

```
        errorMsg "Error getting oamStatus on $chassisId  
$cardId $portId"  
    }  
    ixPuts "Local oamVersion:[oamStatus cget -oamVersion]"  
    ixputs "Local discoveryStatus:[oamStatus cget  
-discoveryStatus]"  
  
    if {[oamStatus getRemoteStatus]} {  
        errorMsg "Error getting oamStatus on $chassisId  
$cardId $portId"  
    }  
    ixPuts "Remote oamVersion:[oamStatus cget -oamVersion]"  
    ixputs "Remote discoveryStatus:[oamStatus cget  
-discoveryStatus]"  
}
```

SEE ALSO*oamStatus*

NAME - oamRemoteInformationTlv

oamRemoteInformationTlv - implements one type of OAM Information TLV.

SYNOPSIS

oamRemoteInformationTlv *sub-command options*

DESCRIPTION

The **oamRemoteInformationTlv** command implements one type of OAM Information PDU. Local and remote information is used in the discovery process.

STANDARD OPTIONS

length

Read only. Set to 0. Indicates the length (in octets) of this TLV_tuple.

**enableLinkEvents
*true/false***

Enable to interpret link events. (*default = false*)

**enableOamPassive
Mode *true/false***

DTE configured in active (*true*) or passive mode. (*default = false*)

**enableRemote
Loopback *true/false***

Enable OAM remote loopback mode. (*default = false*)

**enableUnidirectional
*true/false***

OAM provides an OAM PDU-based mechanism to notify the remote DTE when one direction of a link is non-operational and therefore data transmission is disabled. The ability to operate a link in a unidirectional mode for diagnostic purposes supports failure detection and notification. (*default = false*)

**enableVariable
Retrieval *true/false***

Enable variable retrieval (see *oamVariableRequest*). (*default = false*)

maxPduSize

11-bit field which represents the largest OAM PDU, in octets, supported by the DTE. This value is compared to the remote's Maximum PDU Size and the smaller of the two is used. (*default = 0*)

multiplexerAction

(*default = oamMultiplexerActionForward*)

Option	Value	Usage
oamMultiplexerActionForward	0	sends on the request over the wire
oamMultiplexerActionDiscard	1	discards the request

parserAction

(*default = oamParserActionForward*)

Option	Value	Usage
oamParserActionForward	0	lower layer forwards request to upper layer
oamParserActionLoopback	1	lower layer sends back request
oamParserActionDiscard	2	lower layer discards request
oamParserActionInvalid	3	parser action is invalid

A***oamRemoteInformationTlv***

revision The current revision of the Information TLV. The value of this field shall start at zero and be incremented each time something in the Information TLV changes. (*default = 0*)

oamVersion *Read only.* Set to 1.

oui Organization unique identifier. 3 hex bytes. (*default = '00 00 00'*)

vendorSpecific Information An unspecified list of hex bytes. May be used to differentiate a vendor's product models/versions. (*default = '00 00 00 00'*)

COMMANDS The **oamRemoteInformationTlv** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamRemoteInformationTlv setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES See example code under *oamHeader*.

SEE ALSO *oamHeader, oamInformation, oamLocalInformationTlv, oamOrganizationSpecificTlv*

NAME - oamStatus

oamStatus - allows polling the OAM status information.

SYNOPSIS	<code>oamStatus sub-command options</code>
DESCRIPTION	The oamStatus command is used to poll the OAM status information for both local and remote clients. The OAM must first be enabled on the port using <i>oamPort</i> .
<hr/>	
STANDARD OPTIONS	
discoveryStatus	<i>Read only</i> . Displays the OAM discovery status. (<i>default = unsatisfied</i>)
version	<i>Read only</i> . Displays the OAM version status. (<i>default = 0</i>)
informationRevision	<i>Read only</i> . Displays the OAM information revision status. (<i>default = 0</i>)
multiplexerAction	<i>Read only</i> . Displays the OAM multiplexer action status. (<i>default = forward</i>)
parserAction	<i>Read only</i> . Displays the OAM parser action status. (<i>default = invalid</i>)
mode	<i>Read only</i> . Displays the OAM mode status. (<i>default = passive</i>)
unidirectionalSupport	<i>Read only</i> . Displays the OAM unidirectionalSupport status. (<i>default = not supported</i>)
loopback	<i>Read only</i> . Displays the OAM loopback status. (<i>default = not supported</i>)
linkEvents	<i>Read only</i> . Displays the OAM link events status. (<i>default = not supported</i>)
mibVars	<i>Read only</i> . Displays the OAM variable retrieval status. (<i>default = not supported</i>)
pduSize	<i>Read only</i> . Displays the OAM PDU size. (<i>default = 0</i>)
oui	<i>Read only</i> . Displays the OAM organization unique identifier status. (<i>default = '00 00 00'</i>)
vendorSpecificInformation	<i>Read only</i> . Displays the OAM vendor specific information. (<i>default = '00 00 00 00'</i>)
sourceMacAddress	<i>Read only</i> . Displays the OAM source MAC address.
<hr/>	
COMMANDS	The oamStatus command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
<hr/>	
oamStatus setDefault	Sets to IxTclHal default values for all configuration options.

A***oamStatus***

oamStatus get chasID cardID portID

Retrieves the OAM status for both local and remote client from the port.

oamStatus getLocalStatus

Retrieves the OAM local status.

oamStatus getRemoteStatus

Retrieves the OAM remote status.

EXAMPLES

See examples under the *oamPort* command.

SEE ALSO

oamPort

NAME - oamSummaryTlv

oamSummaryTlv - implements one type of OAM Event Notification PDU.

SYNOPSIS	<i>oamSummaryTlv sub-command options</i>
DESCRIPTION	<p>The oamSummaryTlv command implements one type of OAM Event Notification PDU.</p> <p>The Errored Frame Seconds Summary Event TLV counts the number of errored frame seconds that occurred during the specified period. The period is specified by a time interval. This event is generated if the number of errored frame seconds is equal to or greater than the specified threshold for that period. An errored frame second is a one second interval wherein at least one frame error was detected.</p> <p>This event is generated at the end of the event window rather than when the threshold is crossed.</p>
<hr/>	
STANDARD OPTIONS	
length	<i>Read only.</i> Set to 18 (0x12). This one-octet field indicates the length (in octets) of this TLV_tuple.
frameSeconds	This two-octet field indicates the number of errored frame seconds in the period, encoded as a 16-bit unsigned integer. (<i>default = 0</i>)
timestamp	This two-octet field indicates the time reference when the event was generated, in terms of 100 ms intervals, encoded as a 16-bit unsigned integer. (<i>default = 0</i>)
window	<p>This two-octet field indicates the duration of the period in terms of 100 ms intervals, encoded as a 16-bit unsigned integer. (<i>default = 0</i>)</p> <p>Lower bound: 10 seconds Upper bound: 900 seconds</p>
threshold	<p>This two-octet field indicates the number of errored frame seconds in the period is required to be equal to or greater than in order for the event to be generated, encoded as a 16-bit unsigned integer. (<i>default = 0</i>)</p> <p>Lower bound: zero errored seconds Upper bound: unspecified</p>
errorRunningTotal	This four-octet field indicates the sum of errored frame seconds that have been detected since the OAM sublayer was reset. (<i>default = 0</i>)
eventRunningTotal	This four-octet field indicates the number of Errored Frame Seconds Summary Event TLVs that have been generated since the OAM sublayer was reset, encoded as a 32-bit unsigned integer. (<i>default = 0</i>)

COMMANDS

The **oamSummaryTlv** command is invoked with the following sub-commands.
If no sub-command is specified, returns a list of all sub-commands available.

oamSummaryTlv setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, *oamEventNotification*, *oamOrganizationSpecificTlv*,
oamLocalInformationTlv, *oamRemoteInformationTlv*, *oamSymbolPeriodTlv*,
oamFrameTlv, *oamFramePeriodTlv*

NAME - oamSymbolPeriodTlv

oamSymbolPeriodTlv - implements one type of OAM Event Notification PDU.

SYNOPSIS

oamSymbolPeriodTlv *sub-command options*

DESCRIPTION

The **oamSymbolPeriodTlv** command implements one type of OAM Event Notification PDU. It counts the number of symbol errors that occurred during the specified period. The period is specified by the number of symbols that can be received in a time interval on the underlying physical layer. This event is generated if the symbol error count is equal to or greater than the specified threshold for that period.

STANDARD OPTIONS

length

Read only. This one-octet field set to 40 (0x28) Indicates the length (in octets) of this TLV_tuple.

symbols

This eight-octet field indicates the number of symbol errors in the period, , encoded as a 64-bit unsigned integer. (*default = 0*)

timestamp

This two-octet field indicates the time reference when the event was generated, in terms of 100 ms intervals, encoded as a 16-bit unsigned integer. (*default = 0*)

window

This eight-octet field indicates the number of symbols in the period, encoded as a 64-bit unsigned integer. (*default = 0*)

Lower bound: the number of symbols in one second for the underlying physical layer.

Upper bound: the number of symbols in one minute for the underlying physical layer.

threshold

This eight-octet field indicates the number of errored symbols in the period is required to be equal to or greater than in order for the event to be generated, encoded as a 64-bit unsigned integer. (*default = 0*)

Lower bound: zero symbol errors.

Upper bound: unspecified.

errorRunningTotal

This eight-octet field indicates the sum of symbol errors since the OAM sublayer was reset. (*default = 0*)

eventRunningTotal

This four-octet field indicates the number of Errored Symbol Period Event TLVs that have been generated since the OAM sublayer was reset, encoded as a 32-bit unsigned integer. (*default = 0*)

COMMANDS

The **oamSymbolPeriodTlv** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamSymbolPeriodTlv setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, oamEventNotification, oamOrganizationSpecificTlv, oamLocalInformationTlv, oamRemoteInformationTlv, oamSymbolPeriodTlv, oamFrameTlv, oamFramePeriodTlv

NAME - oamVariableRequest

oamVariableRequest - the OAM PDU type *Variable Request* allows querying MIB variables.

SYNOPSIS

oamVariableRequest *sub-command options*

DESCRIPTION

The **oamVariableRequest** command implements the OAM PDU type *Variable Request*.

It is used to query MIB variables, using data structures called Variable Descriptors. An OAM client may request one or more variables in each Variable Request OAM PDU.

The TLV connected to this command is *oamVariableRequestTlv*.

STANDARD OPTIONS

currentTlvType

Read only. (default = oamVariableRequestEndOfTlv)

Predefined options include:

Option	Value	Usage
oamVariableRequestEndOfTlv	0x00	(default) End of TLV Marker
oamVariableRequest	0x01	Variable Request

COMMANDS

The **oamVariableRequest** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamVariableRequest **setDefault**

Sets to IxTclHal default values for all configuration options.

oamVariableRequest **addTlv *tlvType***

Adds a TLV to OAM Variable Request PDU with TLV type (see *currentTlvType*, above).

oamVariableRequest **delTlv *tlvIndex***

Deletes a TLV from OAM Variable Request with specific Index.

oamVariableRequest **setTlv *tlvIndex***

Sets the configuration of the TLV with the specified Index.

oamVariableRequest **getTlv *tlvIndex***

Gets the configuration of the TLV with the specified Index.

oamVariableRequest **getFirstTlv**

Gets the first TLV from the list of OAM Variable Request PDUs.

oamVariableRequest getNextTlv

Gets the next TLV from the list.

oamVariableRequest clearAllTlvs

Clears all TLVs for the Variable Request PDU.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, oamVariableRequestTlv, oamVariableResponse

NAME - oamVariableRequestTlv

oamVariableRequestTlv - implements one type of OAM Variable Request PDU.

SYNOPSIS

oamVariableRequestTlv *sub-command options*

DESCRIPTION

The **oamVariableRequestTlv** command implements one type of OAM Variable Request PDU.

STANDARD OPTIONS

branch

One-byte hex number. (*default* - 0x07)

Branch of data within the Management Information Base (MIB)

Variable Branches may reference attributes, objects or packages. If an object or package is referenced, only the attributes within the object or package shall be found within the Variable Container.

leaf

Two-byte hex number. (*default* - '00 02')

Sub-branch of data within the Management Information Base (MIB)

COMMANDS

The **oamVariableRequestTlv** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamVariableRequestTlv setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, oamVariableRequest

NAME - oamVariableResponse

oamVariableResponse - the OAM PDU type *Variable Response* allows returning the results of querying MIB variables.

SYNOPSIS

oamVariableResponse *sub-command options*

DESCRIPTION

The **oamVariableResponse** command implements the OAM PDU type *Variable Response*. It is used to return the results of queries of MIB variables, using data structures called Variable Containers. Each returned Variable Container resides within a single Variable Response OAM PDU. If a Variable Container does not fit within a Variable Response OAM PDU, an error code is returned.

In returning requested variables, an OAM client generates at least one and perhaps additional Variable Response OAM PDUs per received Variable Request OAM PDU.

The TLV connected to this command is *oamVariableResponseTlv*.

STANDARD OPTIONS

currentTlvType

Read only. (default = oamVariableResponseEndOfTlv)

Predefined options include:

Option	Value	Usage
oamVariableResponseEndOfTlv	0x00	(default) End of TLV Marker
oamVariableResponse	0x01	Variable Response

COMMANDS

The **oamVariableResponse** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamVariableResponse **addTlv tlvType**

Adds a TLV to OAM Variable Response PDU with TLV type (see *currentTlvType*, above).

oamVariableResponse **delTlv tlvIndex**

Deletes a TLV from OAM Variable Response with specific Index.

oamVariableResponse **setTlv tlvIndex**

Sets the configuration of the TLV with the specified Index.

oamVariableResponse **getTlv tlvIndex**

Gets the configuration of the TLV with the specified Index.

oamVariableResponse **getFirstTlv**

Gets the first TLV from the list of OAM Variable Response PDUs.

oamVariableResponse getNextTlv

Gets the next TLV from the list.

oamVariableResponse clearAllTlvs

Clears all TLVs for the Variable Response PDU.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, oamVariableResponseTlv, oamVariableRequest

NAME - oamVariableResponseTlv

oamVariableResponseTlv - implements one type of OAM Variable Response PDU.

SYNOPSIS

oamVariableResponseTlv *sub-command options*

DESCRIPTION

The **oamVariableResponseTlv** command implements one type of OAM Variable Response PDU.

STANDARD OPTIONS

branch

The one-octet *branch* field for the specific attribute, package or object being returned. (*default* = 0x07)

Note: Only attributes are supported in the current implementation.

If an object or package is referenced, only the attributes within the object or package shall be found within the Variable Container.

leaf

The two-octet *leaf* field for the specific attribute, package or object being returned. (*default* = '00 02')

**enableIndication
true/false**

When *true*, the variable *indicationValue* is present and there is no *value* field.

When *false*, the variable *width* represents the length of the *value* field in octets. (*default* = *false*)

indicationValue

Variable indication. (*default* = 0x04) See *width*, below.

width

The width of the value. If *enableIndication* is *true*, then *width* can be from 0 to 128. (*default* = 4)

This field either contains the actual width of the attribute or an *indicationValue* providing information as to the reason this particular attribute could not be returned.

When bit 7 = 1, bits 6:0 represent an *indicationValue*. There is no *value* field when bit 7 = 1.

When bit 7 = 0, bits 6:0 represent the length of the *value* field in octets. An encoding of 0x00 equals 128 octets. All other encodings represent actual lengths.

value

If *enableIndication* is *true*, then this can be the value of *width* size hex number. (*default* = '00 00 00 00')

If the *width* field contains a width value, the fourth field is the *value* field, which contains the attribute. This field may be up to 128 octets in length. Octets of the attribute are ordered most significant first, followed by each successive octet.

If the *width* field contains an *indicationValue*, the *value* field does not exist.

COMMANDS

The **oamVariableResponseTlv** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

oamVariableResponseTlv setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example code under *oamHeader*.

SEE ALSO

oamHeader, *oamVariableResponse*

NAME - opticalDigitalWrapper

opticalDigitalWrapper - enable the use of optical digital wrapper.

SYNOPSIS

opticalDigitalWrapper *sub-command options*

DESCRIPTION

The *opticalDigitalWrapper* command is used to enable the presence of the optical digital wrapper. This feature is only available for certain port types; this may be tested through the use of the *port isvalidFeature... portFeatureFec* command. None of the overhead bytes may be modified at this point, only the Forward Error Correction (FEC) feature may be changed.

STANDARD OPTIONS

enableFec <i>true/false</i>	Enables the use of the optical digital wrapper and the inclusion of FEC. (<i>default = false</i>)
enableStuffing <i>true / false</i>	Enables the use of fixed stuffing in 10G LAN mode. Additional overhead bytes are added into the overhead and the clock rate is higher to carry the same date. With Fixed stuffing, the line rate is 11.09573 Gb/s, as opposed to 11.04911 Gb/s. (<i>default = false</i>)
payloadType	The data type that is being simulated in the payload area of the SONET frame.

Option	Value	Usage
<i>optDigWrapperPayloadType02</i>	0x02	ASY STM-N
<i>optDigWrapperPayloadType03</i>	0x03	(<i>default</i>) BIT SYN STM-N
<i>optDigWrapperPayloadType04</i>	0x04	ATM
<i>optDigWrapperPayloadType05</i>	0x05	GFP
<i>optDigWrapperPayloadType10</i>	0x10	10-bit STR with O.T.
<i>optDigWrapperPayloadType11</i>	0x11	11-bit STR with O.T.
<i>optDigWrapperPayloadTypeFE</i>	0xFE	FE-PRBS

COMMANDS

The *opticalDigitalWrapper* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands.

opticalDigitalWrapper **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *opticalDigitalWrapper* command.

opticalDigitalWrapper **config** *option value*

Modify the configuration options. If no *option* is specified, returns a list describing all of the available MPLS label options (see STANDARD OPTIONS) for port.

opticalDigitalWrapper get *chassisID cardID portID*

Gets the current configuration of the specified port. Call this command before calling *opticalDigitalWrapper cget option* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

opticalDigitalWrapper set *chassisID cardID portID*

Sets the configuration for the indicated port, reading the configuration option values set by the *opticalDigitalWrapper config option value* command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

opticalDigitalWrapper setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under the *fecError* command.

SEE ALSO

fecError

NAME - packetGroup

packetGroup - configure the Packet Group parameters.

SYNOPSIS	packetGroup <i>sub-command options</i>
DESCRIPTION	<p>The packetGroup command is used to configure the parameters for Packet Groups. Packet groups are given unique IDs within which metrics such as minimum, maximum and average latency for every incoming frame is calculated by the hardware in real-time.</p> <p>An additional feature available on some port types is the ability to collect latency measurements per packet group. The availability of this feature for a given port can be tested using the <i>port isValidFeature... portFeatureRxLatencyBin</i>. The port must be configured for wide packet groups (the <i>port's receiveMode</i> includes the <i>portRxModeWidePacketGroup</i> bit); the availability of this mode may be tested with <i>port isValidFeature... portFeatureRxWidePacketGroups</i>. No configuration is necessary on the transmit port; only the receive port must be configured to receive latency bin operation. This feature is enabled on the receive port with the <i>enableLatencyBins</i> option.</p> <p>The latency measurements for each packet group may be collected in a set up to 16 continuous latency buckets. The first bucket always starts at 0 and the last bucket always ends at the maximum latency. The <i>packetGroup</i> interface allows for the specification of up to 15 time dividers between latency bins. For example, to specify five latency buckets for:</p> <ul style="list-style-type: none"> • 0 - 0.70ms • 0.70ms - 0.72ms • 0.72ms - 0.74ms • 0.74ms - 0.76ms • 0.76ms - max <p>one programs four dividing times:</p> <ul style="list-style-type: none"> • 0.70ms • 0.72ms • 0.74ms • 0.76ms <p>This is done through the <i>latencyBins</i> option. No other setup is required for the receive side port. The latency statistics per latency bin are obtained through the use of the <i>packetGroupStats</i> command.</p> <p>An additional feature available on some port types is the ability to measure latency as it varies over time. The availability of this feature for a given port can be tested using the <i>port isValidFeature... portFeatureRxTimeBin</i>. The port must be configured for wide packet groups (the <i>port's receiveMode</i> includes the <i>portRxModeWidePacketGroup</i> bit); the availability of this mode may be tested with <i>port isValidFeature... portFeatureRxWidePacketGroups</i>. No configuration is necessary on the transmit port; only the receive port must be configured to receive time bin operation. This feature is enabled on the receive port with the <i>enableTimeBins</i> option.</p> <p>The latency over time for each packet group may be collected for a number of evenly spaced time periods, as indicated by the <i>numTimeBins</i> and</p>

timeBinDuration options. The number of packet groups used per time bin must also be specified in the *numPgidPerTimeBin* option.

The product of *numPgidPerTimeBin* (which must be a power of 2) and the next higher power of 2 of the *numTimeBins* must be less than the total number of packet group IDs available for the port when not in time bin mode.

The latency statistics per time bin are obtained through the use of the *packetGroupStats* command.

Time bins and latency bins may be used at the same time.

Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in *atmHeader*. The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command is with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

STANDARD OPTIONS

allocateUdf *true / false* Assigns one of the User-Defined Fields for use with the Sequence Number.
(*default = true*)

delayVariationMode Selects Delay Variation measurement mode (under Latency/Jitter Measurement). This only is available when *measurementMode* is set to *packetGroupModeDelayVariation*.

Note: When Delay Variation mode is selected, then under Sequence Checking the only available option is Threshold Sequence Checking.

Delay Variation options include::

Option	Value	Usage
<i>delayVariationWith SequenceErrors</i>	0	(<i>default</i>) delay variation with sequence errors
<i>delayVariationWith LatencyMinMax</i>	1	delay variation with latency min/max
<i>delayVariationWith LatencyAvg</i>	2	delay variation with latency average

enable128kBinMode *true / false* If *true*, then the length of the packet group ID field is increased to 17 bits.
(*default = false*)

enableGroupIdMask *true / false* Enables the use of the *groupId* mask. (*default = false*)

enableInsertPgid *true / false* Enables inserting the PGID into the packet. (*default = false*)

enableLastBitTime Stamp *true/false* If true, enables selection of last bit time stamp. If false, the first bit time stamp is used. (*default = false*)

enableLatencyBins *true / false* Enables the use of latency bins on receive. (*default = false*)

enableReArmFirstTimeStamp	Enables the use of RE Arm first time stamp.																		
enableRxFilter <i>true / false</i>	Enables the use of the <i>headerFilterMask</i> mask. (<i>default = false</i>)																		
enableSignatureMask <i>true / false</i>	Enables the use of the <i>signatureMask</i> mask. (<i>default = false</i>)																		
enableTimeBins <i>true / false</i>	Enables the use of time bins on receive. (<i>default = false</i>)																		
groupId	Unique value used to identify one packet group for another. Up to 57344 different packet groups may be defined. (<i>default = 0</i>)																		
groupIdMask	A two-byte mask applied to the group ID. Bits which are ‘1’ in the mask are set to ‘0’ in the received group ID. (<i>default = 0</i>)																		
groupIdMode	This option provides a convenience mechanism for setting the <i>groupIdOffset</i> and <i>groupIdMask</i> .																		
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Option</th> <th style="text-align: center;">Value</th> <th style="text-align: center;">Usage</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;"><i>packetGroupCustom</i></td> <td style="text-align: center;">0</td> <td>(<i>default</i>) The offset and mask are set in <i>groupIdOffset</i> and <i>groupIdMask</i>.</td> </tr> <tr> <td style="text-align: center;"><i>packetGroupDscp</i></td> <td style="text-align: center;">1</td> <td>The offset and mask are set to the DSCP location; that is, <i>groupIdOffset</i>=14 and <i>groupIdMask</i>=FF03.</td> </tr> <tr> <td style="text-align: center;"><i>packetGroupIpv6TrafficClass</i></td> <td style="text-align: center;">2</td> <td>The offset and mask are set to the destination IPv6 address' traffic class; that is, <i>groupIdOffset</i>=14 and <i>groupIdMask</i>=FD3F.</td> </tr> <tr> <td style="text-align: center;"><i>packetGroupMplsExp</i></td> <td style="text-align: center;">3</td> <td>The offset and mask are set to the MPLS label's Experimental field; that is, <i>groupIdOffset</i>=16 and <i>groupIdMask</i>=F1FF.</td> </tr> <tr> <td style="text-align: center;"><i>packetGroupSplit</i></td> <td style="text-align: center;">4</td> <td>The offset and mask are set in to split PGIDs.</td> </tr> </tbody> </table>		Option	Value	Usage	<i>packetGroupCustom</i>	0	(<i>default</i>) The offset and mask are set in <i>groupIdOffset</i> and <i>groupIdMask</i> .	<i>packetGroupDscp</i>	1	The offset and mask are set to the DSCP location; that is, <i>groupIdOffset</i> =14 and <i>groupIdMask</i> =FF03.	<i>packetGroupIpv6TrafficClass</i>	2	The offset and mask are set to the destination IPv6 address' traffic class; that is, <i>groupIdOffset</i> =14 and <i>groupIdMask</i> =FD3F.	<i>packetGroupMplsExp</i>	3	The offset and mask are set to the MPLS label's Experimental field; that is, <i>groupIdOffset</i> =16 and <i>groupIdMask</i> =F1FF.	<i>packetGroupSplit</i>	4	The offset and mask are set in to split PGIDs.
Option	Value	Usage																	
<i>packetGroupCustom</i>	0	(<i>default</i>) The offset and mask are set in <i>groupIdOffset</i> and <i>groupIdMask</i> .																	
<i>packetGroupDscp</i>	1	The offset and mask are set to the DSCP location; that is, <i>groupIdOffset</i> =14 and <i>groupIdMask</i> =FF03.																	
<i>packetGroupIpv6TrafficClass</i>	2	The offset and mask are set to the destination IPv6 address' traffic class; that is, <i>groupIdOffset</i> =14 and <i>groupIdMask</i> =FD3F.																	
<i>packetGroupMplsExp</i>	3	The offset and mask are set to the MPLS label's Experimental field; that is, <i>groupIdOffset</i> =16 and <i>groupIdMask</i> =F1FF.																	
<i>packetGroupSplit</i>	4	The offset and mask are set in to split PGIDs.																	
groupIdOffset	The offset, within the packet, of the group id value. (<i>default = 52</i>)																		
headerFilter	A set of 16 bytes used to match the header of packets to be considered for signature masking. (<i>default = {00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00}</i>)																		
headerFilterMask	A mask to be applied to the <i>headerFilter</i> . Bits which are ‘1’ are ignored in the match. (<i>default = {00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00}</i>)																		
ignoreSignature <i>true / false</i>	In receive mode, the signature field is not matched and all packets are counted. (<i>default = false</i>)																		
insertSequence Signature <i>true / false</i>	Inserts a sequence signature into the packet as indicated by <i>signatureOffset</i> , <i>signatureValue</i> , <i>groupIdOffset</i> , <i>signatureNumberOffset</i> and <i>allocateUdf</i> . (<i>default = false</i>)																		
insertSignature <i>true/false</i>	Inserts the packet group signature into the transmitted stream. (<i>default = false</i>) Note: For calculating latency values need to configure stream config –fir true .																		

latencyBinList

If *enableLatencyBins* is *true*, this TCL array of floating point numbers, each of which is expressed in milliseconds, indicates the dividing line between latency bins. As per the discussion at the head of this command, there is one fewer dividing times than latency bins. The first bin always starts at 0 and the last bin always ends at the maximum possible latency. The list is sorted before use. There must not be any duplicate values. (*default* = “”)

latencyControl

Defines the mechanism used to calculate latency. Possible values include:

Option	Value	Usage
<i>cutThrough</i>	0x01	(<i>default</i>) first data bit in to first data bit out
<i>storeAndForward</i>	0x03	last data bit in to first data bit out
<i>storeAndForwardPreamble</i>	0x05	last data bit in to first preamble out
<i>interArrivalJitter</i>	0x07	inter-arrival jitter This selection automatically activates <i>measurementMode</i> inter-arrival time mode.
<i>firstInLastOut</i>	0x08	first in last out
<i>lastInLastOut</i>	0x09	last in last out

maxRxGroupId

Read only. Displays the maximum number of PGIDs available for the port based on the receive side configuration.

measurementMode

Defines the measurement mode used to calculate latency. Possible values include:

Option	Value	Usage
<i>packetGroupModeLatency</i>	0	(<i>default</i>) latency mode
<i>packetGroupModeInterArrivalTime</i>	1	inter-arrival time mode Note: Requires <i>latencyControl</i> to be set to <i>interArrivalJitter</i> . On the other hand, simply setting <i>latencyControl</i> to <i>interArrivalJitter</i> automatically selects this value for <i>measurementMode</i> .
<i>packetGroupModeDelayVariation</i>	2	delay variation mode Selecting this mode automatically enables <i>delayVariationMode</i> option.

multiSwitchedPath Mode

Two alternatives exist for the manner in which time stamps are used when *sequenceCheckingMode* is set to *seqMultiSwitchedPath*:

Option	Value	Usage
<i>seqSwitchedPathPGID</i>	0	(<i>default</i>) Time stamps are used to hold the time stamp of the first packet received for each packet group ID.
<i>seqSwitchedPathDuplication</i>	1	Time stamps are used to hold the first detected packet duplication. Note: Not available when in Delay Variation mode.

numPgidPerTimeBin

If *enableTimeBins* is *true*, this is used as the number of packet group IDs to be received for each time bin defined by *timeBinDuration*. Note that this value must be a power of 2; for example, 1, 2, 4, 8, 16, .. Note that the product of this number and the next higher power of 2 of the *numTimeBins* option must not exceed the

total number of packet group IDs available for the port if *enableTimeBins* were *false*. (*default* = 32)

numTimeBins

If *enableTimeBins* is true, this is used as the number of distinct time bins to collect latency over, per packet group ID. The range of legal values is from 1 to 2048. The length of all time bins is dictated by *timeBinDuration*. Note that the product of the next higher power of 2 of this number and *numPgidPerTimeBin* option must not exceed the total number of packet group IDs available for the port if *enableTimeBins* were *false*. (*default* = 10)

preambleSize

Length of preamble, in bytes, of received frame. (*default* = 8)

seqAdvTracking

If true, allows to track a frame by following five new statistics:

statistics
<i>In Order</i>
<i>Reorder</i>
<i>Duplicate</i>
<i>Late</i>
<i>Lost</i>

seqAdvTrackingLateThreshold

A fixed value that sets a threshold to track the expected sequence value. The Late Threshold value is subtracted from the expected sequence number when the received sequence numbers are less than the late threshold value.

sequenceErrorThreshold

The threshold value used to determine whether a sequence error has occurred. (*default* = 2)

sequenceCheckingMode

The manner in which sequence checking is performed.

Option	Value	Usage
<i>seqThreshold</i>	0	(<i>default</i>) Sequence errors are determined by checking sequence numbers against <i>sequenceErrorThreshold</i> .
<i>seqMultiSwitchedPath</i>	1	Sequence checking is performed looking for skipped and duplicate sequence numbers. Check the <i>portFeatureMultiSwitchPacketDetection</i> feature for availability of this choice.
<i>seqAdvTracking</i>	2	Advanced sequence tracking is enabled.

sequenceNumberOffset

The offset within the packet of the sequence number. This is valid only when sequence checking is enabled. (*default* = 44)

signature

In the transmitted packet, the signature uniquely signs the transmitted packet as one destined for packet group filtering on the receive port. On the receive port, the signature is used to filter only those packets that have a matching signature and the minimum, maximum and average latencies are obtained for those packets. (*default* = {08 71 18 05})

signatureMask

A four-byte mask applied to the signature. Bits which are ‘1’ in the mask are ignored. (*default* = {00 00 00 00})

signatureOffset

The offset, within the packet, of the packet group signature. (*default* = 48)

timeBinDuration

If *enableTimeBins* is true, this is the length of each time bin. This value is expressed in nanoseconds. (*default = 1000000*)

COMMANDS

The **packetGroup** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

packetGroup cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **packetGroup** command.

packetGroup config option value

Modify the Packet Group configuration options of the port. If no *option* is specified, returns a list describing all of the available Packet Group options (see STANDARD OPTIONS) for port.

packetGroup get CircuitTx chasID cardID portID [circuitID] streamID

Gets the current configuration of the stream with id *streamID* in the circuit with *circuitID* on port *portID*, card *cardID*, chassis *chasID* from its hardware.

packetGroup get QueueTx chasID cardID portID [queueID] streamID

Gets the current configuration of the stream with id *streamID* in the queue with *queueID* on port *portID*, card *cardID*, chassis *chasID* from its hardware.

packetGroup get Rx chasID cardID portID

Gets the current receive Packet Group configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Call this command before calling **packetGroup cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

packetGroup get Tx chasID cardID portID streamID [type]

Gets the current transmit Packet Group configuration of the stream with id *portID* on card *cardID*, chassis *chasID*, stream *streamID*.

In the first form, the *queueID* indicates the particular queue for load modules which use multiple queues, such as ATM cards.

In the second form, the type of stream (stream or flow) is selected. One of.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(<i>default</i>) Both streams and flows. This option can be used for ports that do not use flows.
<i>streamSequenceTypeStreams</i>	1	Stream only.
<i>streamSequenceTypeFlows</i>	2	Flow only.

Call this command before calling **packetGroup cget option value** to get the value of the configuration option. specific errors are:

- No connection to a chassis

A

packetGroup

- Invalid port number
- The port is being used by another user
- No stream has been configured for the streamID

packetGroup **setCircuitTx** *chasID cardID portID [circuitID] streamID*

Sets the configuration of the stream with id *streamID* on its circuit *circuitID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL by reading the configuration option values set by the **packetGroup config option value** command.

packetGroup **setDefault**

Sets to IxTclHal default values for all configuration options.

packetGroup **setQueueTx** *chasID cardID portID [queueID] streamID*

Sets the configuration of the stream with id *streamID* on its queue *queueID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL by reading the configuration option values set by the **packetGroup config option value** command.

packetGroup **setRx** *chasID cardID portID*

Sets the receive Packet Group configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **packetGroup config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

packetGroup **setTx** *chasID cardID portID streamID [type]*

Sets the transmit Packet Group configuration of the stream with id *portID* on card *cardID*, chassis *chasID*, stream *streamID* by reading the configuration option values set by the **packetGroup config option value** command.

In the first form, the *queueID* indicates the particular queue for load modules which use multiple queues, such as ATM cards.

In the second form, the type of stream (stream or flow) is selected. One of.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) Both streams and flows. This option can be used for ports that do not use flows.
<i>streamSequenceTypeStreams</i>	1	Stream only.
<i>streamSequenceTypeFlows</i>	2	Flow only.

After calling this command, the Packet Group configuration should be committed to hardware using **stream write** or **ixWriteConfigToHardware** commands. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- No stream has been configured for the streamID

EXAMPLES

```
package require IxTclHal
```

```
# In this example, we'll measure the latency for different frame
# sizes through a simple switch using packet groups.
# Latency bins will also be retrieved
# Port 1 is used to transmit three streams, each with a packet
# group signature and packet group ID equal to the stream ID.
# 100,000 packets are transmitted by each stream
# Port 2 is used to received the data using Packet Group Mode.
# A short stream is transmitted from this port to the switch in
# order to get the switch to 'learn' its MAC address
# Separate sections are included for latency bins and time bins

# Connect to chassis and get chassis ID
set host techpubs-400
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assumes that card 1 is a 10/100 card with both ports connected
# to a simple L2 switch
set card 1
set txPort 1
set rxPort 2

# Useful port lists
set portList [list [list $chas $card $txPort] \
                    [list $chas $card $rxPort]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

set p1MAC [list 00 00 00 01 01 01]
set p2MAC [list 00 00 00 01 01 02]

set numFrames 4
set numBursts 240
set minSize 64
set maxSize 1024
set stepSize 64

set lb [list 0.70 0.72 0.74 0.76]
```

```

# Need to clear time stamps
if [ixClearTimeStamp portList] {
    ixPuts $::ixErrorInfo
}

ixPuts "Testing $chas:$card:$txPort -> $chas:$card:$rxPort"

# Set up Transmit Port
port setFactoryDefaults $chas $card $txPort
port config -autonegotiate true

# stream: from port 1 MAC to port 2 MAC, stream size as per array
# Make sure to insert time stamp (fir)
set streamID 1

stream setDefault
stream config -numFrames      $numFrames
stream config -numBursts      $numBursts
stream config -sa              $p1MAC
stream config -da              $p2MAC
stream config -fir             true
stream config -dma             stopStream
stream config -frameSizeType  sizeIncr
stream config -frameSizeMIN   $minSize
stream config -frameSizeMAX   $maxSize
stream config -frameSizeStep  $stepSize
stream config -enableIbg      false
stream config -enableIsg      false
stream config -rateMode       usePercentRate
stream config -percentPacketRate 0.04512

udf setDefault
udf config -enable           true
udf config -counterMode     udfCounterMode
udf config -continuousCount  false
udf config -initval          0
udf config -repeat           4
udf config -udfSize          c16
udf config -offset            52

if [udf set 1] {
    errorMsg "Error in udf set"
    return "FAIL"
}
if [stream set $chas $card $txPort $streamID] {
    errorMsg "Error in stream set"
    return "FAIL"
}

# Make sure to insert a PG signature and streamID = PGID
packetGroup setDefault
packetGroup config -insertSignature true
if [packetGroup setTx $chas $card $txPort $streamID] {
    errorMsg "error in packetGroup setTx (1)"
    return "FAIL"
}

# Set up Receive Port
port setFactoryDefaults $chas $card $rxPort
port config -autonegotiate true

# the port must be in packet group mode
port config -receiveMode $::portRxModeWidePacketGroup

```

```

if [port set $chassis $card $rxPort] {
    errorMsg "Error in port set"
    return "FAIL"
}

# Set up receive packet group mode (store and forward)
packetGroup setDefault
packetGroup config -latencyControl      storeAndForward

#####
#
# Latency Bin sample
#
#####

# set the latency bin list
packetGroup config -enableLatencyBins true
packetGroup config -latencyBinList      $lb

if [packetGroup setRx $chassis $card $rxPort] {
    errorMsg "Error in packetGroup setRx"
    return "FAIL"
}

# Let the hardware know about the two ports
ixWriteConfigToHardware portList

if [packetGroup getRx $chassis $card $rxPort] {
    errorMsg "Error in packetGroup getRx"
    return "FAIL"
}

# Wait for changes to take affect and make sure links are up
after 1000
ixCheckLinkState portList

# Start the packet groups on the receive port
# and then the transmit from the transmit port

ixPuts "Starting packet groups"
ixStartPortPacketGroups $chassis $card $rxPort

ixPuts "Starting port transmit"
ixStartPortTransmit      $chassis $card $txPort

after 1000

# and then wait for things to be done
ixPuts "Waiting for transmit done"
ixCheckPortTransmitDone $chassis $card $txPort

ixPuts "Stopping packet groups"
ixStopPortPacketGroups $chassis $card $rxPort

# Now get the statistics back
# First a get for all of the packet groups
if [packetGroupStats get $chassis $card $rxPort 0 16384] {
    errorMsg "Error in packetGroupStats get"
    return "FAIL"
}
set numGroups [packetGroupStats cget -numGroups]

```

```

set numRxLatencyBins [packetGroupStats cget -numLatencyBins]
ixPuts "# received latency bins = $numRxLatencyBins"

ixPuts "PGID      LBin      FrameSz #          MinLat  MaxLat  AvgLat
byRate   frRate   stdDev   "
ixPuts "----- ----- ----- ----- ----- ----- ----- -
----- ----- ----- "
for {set i 0} {$i < $numFrames} {incr i} {

    packetGroupStats getGroup $i
    set totalFrames [packetGroupStats cget -totalFrames]

    ixPuts -nonewline "$i\t"
    ixPuts -nonewline "All\t"
    ixPuts -nonewline "[expr ($i+1) * 64]\t"
    ixPuts -nonewline "$totalFrames\t"
    ixPuts -nonewline [packetGroupStats cget -minLatency]
    ixPuts -nonewline "\t"
    ixPuts -nonewline [packetGroupStats cget -maxLatency]
    ixPuts -nonewline "\t"
    ixPuts -nonewline [packetGroupStats cget -averageLatency]
    ixPuts -nonewline "\t"
    ixPuts -nonewline [packetGroupStats cget -byteRate]
    ixPuts -nonewline "\t"
    ixPuts -nonewline [packetGroupStats cget -frameRate]
    ixPuts -nonewline "\t"
    ixPuts           [packetGroupStats cget -standardDeviation]

    for {set latencyBin 0} {$latencyBin < $numRxLatencyBins} \
        {incr latencyBin} \
    {
        if {$latencyBin == 0} {
            if [packetGroupStats getFirstLatencyBin] {
                errorMsg "Error in packetGroupStats
getFirstLatencyBin"
                return "FAIL"
            }
        } else {
            if [packetGroupStats getNextLatencyBin] {
                errorMsg "Error in packetGroupStats
getNextLatencyBin"
                return "FAIL"
            }
        }
        set numLatencyFrames [latencyBin cget -numFrames]

        if [packetGroupStats getLatencyBin $latencyBin] {
            errorMsg "Error in packetGroupStats getLatencyBin"
            return "FAIL"
        }

        ixPuts -nonewline "\t"
        ixPuts -nonewline "$latencyBin\t\t"
        ixPuts -nonewline "$numLatencyFrames\t"
        ixPuts -nonewline [latencyBin cget -minLatency]
        ixPuts -nonewline "\t"
        ixPuts -nonewline [latencyBin cget -maxLatency]
        ixPuts -nonewline "\t\t"
        ixPuts -nonewline [latencyBin cget -byteRate]
        ixPuts -nonewline "\t"
        ixPuts           [latencyBin cget -frameRate]
    }
}

```

```

}

#####
#
# Time Bin sample
#
#####

set numPgidPerTimeBin      4
set numTimeBins            2
set timeBinDuration        10000000000

# set the time bin options
packetGroup config -enableLatencyBins  false
packetGroup config -enableTimeBins    true
packetGroup config -numPgidPerTimeBin $numPgidPerTimeBin
packetGroup config -numTimeBins       $numTimeBins
packetGroup config -timeBinDuration   $timeBinDuration

if [packetGroup setRx $chassis $card $rxPort] {
    errorMsg "Error in packetGroup setRx"
    return "FAIL"
}

# Let the hardware know about the two ports
ixWriteConfigToHardware portList

if [packetGroup getRx $chassis $card $rxPort] {
    errorMsg "Error in packetGroup getRx"
    return "FAIL"
}

# Wait for changes to take affect and make sure links are up
after 1000
ixCheckLinkState portList

# Start the packet groups on the receive port
# and then the transmit from the transmit port

ixPuts "Starting packet groups"
ixStartPortPacketGroups $chassis $card $rxPort

ixPuts "Starting port transmit"
ixStartPortTransmit      $chassis $card $txPort

after 1000

# and then wait for things to be done
ixPuts "Waiting for transmit done"
ixCheckPortTransmitDone $chassis $card $txPort

ixPuts "Stopping packet groups"
ixStopPortPacketGroups $chassis $card $rxPort

# Now get the statistics back
# First a get for all of the packet groups
if [packetGroupStats get $chassis $card $rxPort 0 16384] {
    errorMsg "Error in packetGroupStats get"
    return "FAIL"
}
set numGroups [packetGroupStats cget -numPgidPerTimeBin]

set numRxTimeBins [packetGroupStats cget -numTimeBins]

```

A

packetGroup

```
ixPuts "# received time bins = $numRxTimeBins"
ixPuts "# PGID per time bin = $numGroups"

ixPuts "T-BIN    PGID      FrameSz #          MinLat  MaxLat  AvgLat
byRate   frRate   stdDev   "
ixPuts "----- ----- ----- ----- ----- ----- -----
----- ----- ----- "

for {set t 1} {$t <= $numRxTimeBins} {incr t} {
    for {set i 0} {$i < $numPgIdPerTimeBin} {incr i} {

        packetGroupStats getGroup $i $t
        set totalFrames [packetGroupStats cget -totalFrames]

        ixPuts -nonewline "$t\t"
        ixPuts -nonewline "$i\t"
        ixPuts -nonewline "[expr ($i+1) * 64]\t"
        ixPuts -nonewline "$totalFrames\t"
        ixPuts -nonewline [packetGroupStats cget -minLatency]
        ixPuts -nonewline "\t"
        ixPuts -nonewline [packetGroupStats cget -maxLatency]
        ixPuts -nonewline "\t"
        ixPuts -nonewline [packetGroupStats cget -averageLatency]
        ixPuts -nonewline "\t"
        ixPuts -nonewline [packetGroupStats cget -byteRate]
        ixPuts -nonewline "\t"
        ixPuts -nonewline [packetGroupStats cget -frameRate]
        ixPuts -nonewline "\t"
        ixPuts           [packetGroupStats cget -standardDeviation]
    }
}
```

SEE ALSO

packetGroupStats

NAME - packetGroupStats

packetGroupStats - retrieve statistics associated with a packet group.

SYNOPSIS	packetGroupStats <i>sub-command options</i>
DESCRIPTION	<p>The packetGroupStats command is used to retrieve the statistics associated with packet groups, such as minimum latency, maximum latency and average latency. Some of the statistics are only available on specific types of ports; an attempt to read an unavailable statistic results in an error. Refer to <i>Appendix B</i> of the <i>Ixia Reference Guide</i> for list of which statistics are available.</p> <p>Three sub-commands are used to retrieve the actual statistics.</p> <ul style="list-style-type: none"> • <i>packetGroupStats get chasID cardID portID [fromPGID toPGID]</i>: this fetches a range of statistics for the indicated port. The range is dictated by the <i>fromPGID</i> to the <i>toPGID</i>; if omitted, only PGID 0 is retrieved. • <i>packetGroupStats getGroup index</i>: this fetches the statistics for a PGID that is $\text{PGID} = \text{fromPGID} + \text{index}$, where <i>fromPGID</i> is the value from the last call to <i>packetGroupStats get</i>. That is, <i>index = 0</i> refers to the <i>fromPGID</i> packet group ID. • <i>packetGroupStats setFrameCount index</i>: operates in the same manner as <i>getGroup</i>, with respect to the <i>index</i> parameter. <p>An additional feature available on some port types is the ability to collect latency measurements per packet group. The availability of this feature for a given port can be tested using the <i>port isValidFeature... portFeatureRxLatencyBin</i>. The port must be configured for wide packet groups (the <i>port's receiveMode</i> includes the <i>portRxModeWidePacketGroup</i> bit); the availability of this mode may be tested with <i>port isValidFeature... portFeatureRxWidePacketGroups</i>. (Note: When the port is in PRBS mode, all latency specific stats are removed.)</p> <p>Latency bin dividing times must be set up with the <i>packetGroup</i>'s <i>enableLatencyBins</i>, <i>latencyBinList</i> option. Following a call to <i>packetGroupStats getGroup</i>, the <i>numLatencyBins</i> option is set and the latency bin information is available through calls to <i>getFirstLatencyBin</i>, <i>getNextLatencyBin</i> and <i>getLatencyBin</i>. The latency information is available in the options of the <i>latencyBin</i> command. Note that there is one more latency bin available than the number of dividers set in <i>packetGroup</i>'s <i>latencyBinList</i>, due to the implicit creation of a latency bin from the last divider to the maximum possible latency value.</p> <p>An additional feature available on some port types is the ability to measure latency over time, per packet group. The availability of this feature for a given port can be tested using the <i>port isValidFeature... portFeatureRxTimeBin</i>. The port must be configured for wide packet groups (the <i>port's receiveMode</i> includes the <i>portRxModeWidePacketGroup</i> bit); the availability of this mode may be tested with <i>port isValidFeature... portFeatureRxWidePacketGroups</i>.</p> <p>Time bins must be set up with the <i>packetGroup</i>'s <i>enableTimeBins</i>, <i>numPgidPerTimeBin</i>, <i>numTimeBins</i> and <i>timeBinDuration</i> options. Following a call to <i>packetGroupStats getGroup</i>, the <i>numTimeBins</i>, <i>numPgidPerTimeBin</i> and <i>timeBinDuration</i> options are set. Latency information for a particular time bin</p>

can be obtained by using the additional *timeBin* argument to the *getGroup* and *getGroupFrameCount* sub-commands.

STANDARD OPTIONS

averageLatency	<i>Read-only. 64-bit value.</i> Average latency for all frames of this packet group. Updated after <i>packetGroupStats getGroup</i> command is called. Used for cut-through, store-forward, and inter-arrival statistics.
bigSequenceError	<i>Read-only. 64-bit value.</i> The number of times when the current sequence number minus the previous sequence number is greater than the error threshold. (Also available in PRBS mode, depending on sequence checking settings.)
bitRate	<i>Read-only. 64-bit value.</i> The bit rate for the frames. Note: this value is calculated on the difference between two successive readings; <i>packetGroupStats get</i> must be called at least twice before valid values are obtained. (Also available in PRBS mode.)
byteRate	<i>Read-only. 64-bit value.</i> The byte rate for the frames. Note: this value is calculated on the difference between two successive readings; <i>packetGroupStats get</i> must be called at least twice before valid values are obtained. (Also available in PRBS mode.)
duplicateFrames	<i>Read-only. 64-bit value.</i> The number of duplicate frames when the port is in multi-switched path mode; that is, the <i>sequenceCheckingMode</i> in the <i>packetGroup</i> command is set to <i>seqMultiSwitchedPath</i> . (Also available in PRBS mode, depending on sequence checking settings.)
firstTimeStamp	<i>Read-only. 64-bit value.</i> The time stamp of the first packet received. This is only available when the port is in wide packet group mode; that is, the <i>port's receiveMode</i> includes the <i>portRxModeWidePacketGroup</i> bit. (Also available in PRBS mode.)
frameRate	<i>Read-only. 64-bit value.</i> The frame rate for the frames. Note: this value is calculated on the difference between two successive readings; <i>packetGroupStats get</i> must be called at least twice before valid values are obtained. (Also available in PRBS mode.)
lastTimeStamp	<i>Read-only. 64-bit value.</i> The time stamp of the last packet received. This is only available when the port is in wide packet group mode; that is, the <i>port's receiveMode</i> includes the <i>portRxModeWidePacketGroup</i> bit. (Also available in PRBS mode.)
maxDelayVariation	<i>Read-only. 64-bit value.</i> Maximum Delay Variation. The largest of all delay variations measured for a specific flow from the start of statistic collection.
maxLatency	<i>Read-only. 64-bit value.</i> Maximum latency of all frames of this packet group. Updated after <i>packetGroupStats getGroup</i> command is called. Used for cut-through, store-forward, and inter-arrival statistics.
maxMinDelayVariation	<i>Read-only. 64-bit value.</i> The interval between the Maximum and Minimum Delay Variation. The mathematical subtraction of Min DV from Max DV.

maxminInterval	<i>Read-only.</i> 64-bit value. The interval between the Maximum and Minimum Latency measurement. Updated after <i>packetGroupStats getGroup</i> command is called. Used for cut-through, store-forward, and inter-arrival statistics.
minDelayVariation	<i>Read-only.</i> 64-bit value. Minimum Delay Variation. The smallest of all delay variations measured for a specific flow from the start of statistic collection.
minLatency	<i>Read-only.</i> 64-bit value. Minimum latency of all frames of this packet group. Updated after <i>packetGroupStats getGroup</i> command is called. Used for cut-through, store-forward, and inter-arrival statistics.
numGroups	<i>Read-only.</i> The total number of groups that were actually received.
numLatencyBins	<i>Read-only.</i> The number of latency bins available for a given packet group. Updated after <i>packetGroupStats getGroup</i> command is called. Note that there is one more latency bin available than the number of dividers set in <i>packetGroup</i> 's <i>latencyBinList</i> , due to the implicit creation of a latency bin from the last divider to the maximum possible latency value.
numPgidPerTimeBin	<i>Read-only.</i> The number of packet group IDs that were used for each time bin, if time bins were enabled and configured in the <i>packetGroup</i> command. This is the same as the same named option used in the <i>packetGroup</i> command when time bins were set up for the port.
numTimeBins	<i>Read-only.</i> The number of time bins used, if time bins were enabled and configured in the <i>packetGroup</i> command. This is the same as the same named option used in the <i>packetGroup</i> command when time bins were set up for the port.
prbsBerRatio	<i>Read-only.</i> 64-bit value. Ratio of PRBS errored bits to bits received.
prbsBitsReceived	<i>Read-only.</i> 64-bit value. Number of PRBS bits received.
prbsErroredBits	<i>Read-only.</i> 64-bit value. Number of PRBS errored bits received.
readTimeStamp	<i>Read-only.</i> Reads the timestamp from when the statistics of a packet group were obtained.
reverseSequenceError	<i>Read-only.</i> 64-bit value. The number of times when the current sequence number is less than the previous sequence number. (Also available in PRBS mode, depending on sequence checking settings.)
sequenceGaps	<i>Read-only.</i> 64-bit value. The number of sequence gaps when the port is in multi-switched path mode; that is, the <i>sequenceCheckingMode</i> in the <i>packetGroup</i> command is set to <i>seqMultiSwitchedPath</i> . (Also available in PRBS mode, depending on sequence checking settings.)
smallSequenceError	<i>Read-only.</i> 64-bit value. The number of times when the current sequence number minus the previous sequence number is less than or equal to the error threshold and not negative, or when the current sequence number is equal to the previous sequence number. (Also available in PRBS mode, depending on sequence checking settings.)

standardDeviation	<i>Read-only.</i> 64-bit value. When latency bins are used, this is the standard deviation of the latencies, using each bin's average.
timeBinDuration	<i>Read-only.</i> The time bin duration expressed in nanoseconds, if time bins were enabled and configured in the <i>packetGroup</i> command. This is the same as the same named option used in the <i>packetGroup</i> command when time bins were set up for the port.
totalByteCount	<i>Read-only.</i> 64-bit value. The number of bytes used to calculate the statistics for this packet group.
totalFrames	<i>Read-only.</i> 64-bit value. Total number of frames used to calculate the statistics for this packet group.
totalSequenceError	<i>Read-only.</i> 64-bit value. The sum of the small, bug and reverse sequence errors. (Also available in PRBS mode, depending on sequence checking settings.)

COMMANDS

The *packetGroupStats* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

packetGroupStats cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *packetGroupStats* command.

packetGroupStats clear chasID cardID portID groupIdPairList

Clears all packet group stats for the specified Group ID Pair List, where the list consists of ranges expressed with pairs like this: {{1 100} {300 400} {500 500}}

Note: {500 500} can also be written as {500}: it selects a single value, which is also understood as the range from 500 to 500.

This command clears the PGIDStats for PGIDs in the Group ID Pair List. To clear individual rows of packet groupstats on the port, the port must either be unowned or you must be logged in as the owner of the port. Specific errors are:

- No connection to a chassis
- The port doesn't support the command: ixTcl_unsupportedFeature
- Invalid port number or port is used by someone else:
ixTcl_notAvailable
- Invalid PGIDList
- Failed to execute the clear command
- Any group ID is outside the legal range

packetGroupStats clearTimeStamps chasID cardID portID groupIdPairList

Clears all packet group time stamps for the specified Group ID Pair List, where the list consists of ranges expressed with pairs like this: {{1 100} {300 400} {500 500}}. Specific errors are:

- No connection to a chassis
- The port doesn't support the command: ixTcl_unsupportedFeature
- Invalid port number or port is used by someone else:
ixTcl_notAvailable
- Invalid PGIDList
- Failed to execute the clear command

- Any group ID is outside the legal range

packetGroupStats reArmFirstFrameTimeStamp *chasID cardID portID groupIDPairList*

Gets the timestamp value of the first frame.

packetGroupStats get *chasID cardID portID [fromGroupID toGroupID]*

Gets the current Packet Group statistics on the port. Call this command before calling *packetGroupStats getGroup index* and *packetGroupStats cget option value* to get the value of the configuration option. *fromGroupID* and *toGroupID* are optional and default to 0. In order for this command to succeed, the port must either be unowned, or you must be logged in as the owner of the port. Specific errors are:

- No connection to a chassis
- Invalid port number
- The *fromGroupID* or *toGroupID* is invalid
- Network error between the client and the chassis
- Either group ID is outside the legal range

packetGroupStats getFirstLatencyBin

Following a call to *packetGroupStats getGroup*, a call to this sub-command makes the values associated with the first latency bin available through the *latencyBin* command. Specific errors are:

- No latency bins are available

packetGroupStats getGroup *index [timeBin]*

Gets the Packet Group statistics for this *index* and particular *timeBin*. If *timeBin* is omitted, a value of 1 is used. Before calling this command, packet group statistics must be retrieved using the *packetGroupStats get chasID cardID portID [fromGroupID toGroupID]* command. The *index* is with respect to the range of group IDs retrieved; for example, *index = 0* always refers to the data associated with *fromGroupID* in the last *packetGroupStats get* call. The last index corresponds to (*toGroupID - fromGroupID*). Specific errors are:

- No packet groups are defined
- No packets were counted in packet group *groupID*
- Either *groupID* is outside the legal range

packetGroupStats getGroupFrameCount *index [timeBin]*

Gets the Packet Group statistics for this *index* and *timeBin* and returns the number of frames in the group / time bin. If *timeBin* is omitted, a value of 1 is used. Before calling this command, packet group statistics must be retrieved using the *packetGroupStats get chasID cardID portID [fromGroupID toGroupID]* command. The first group available is always 0 and corresponds to the *fromGroupID* argument to *packetGroupStats get*. The last group is (*toGroupID - fromGroupID*).

packetGroupStats getLatencyBin *lbIndex*

Following a call to *packetGroupStats getFirstLatencyBin*, a call to this sub-command makes the values associated with the latency bin specified by *lbIndex* available through the *latencyBin* command. Specific errors are:

- Invalid latency bins number
- The specified latency bin number does not exist

packetGroupStats *getNextLatencyBin*

Following a call to *packetGroupStats getFirstLatencyBin*, a call to this sub-command makes the values associated with the next latency bin available through the *latencyBin* command. Specific errors are:

- No more latency bins are available

packetGroupStats *setDefault*

Zeros all local statistics in the packet group stat list.

EXAMPLES

See examples under *packetGroup*.

SEE ALSO

packetGroup, latencyBin, stream

NAME - packetGroupThresholdList

packetGroupThresholdList - configure and contain the PGID range threshold values

SYNOPSIS

packetGroupThresholdList *sub-command options*

DESCRIPTION

The *packetGroupThresholdList* command is used to configure and contain the PGID range threshold values.

There is one threshold for each PGID. Use this command to select a range of PGIDs that is configured with the same threshold value.

The fromPGID value defines the start of the range and the toPGID value is the end of the range. To configure just one PGID, use identical 'from' and 'to' values.

STANDARD OPTIONS

enableJitterFilter	true or false to specify that jitter filtering is on or off, default is false
fromPGID	the sequence number of the PGID at the start of the range
toPGID	the sequence number of the PGID at the end of the range
threshold	if enableJitterFilter is OFF, it stands for a number (in nanoseconds) that is the threshold for the Inter-Arrival Time (latency) of a PGID or a range of PGIDs if enableJitterFilter is ON, its units become packets instead of ns
filterWindow	specifies the filter window

Option	Value	Usage
<i>filterWindow81920ns</i>	0	81920
<i>filterWindow163840ns</i>	1	81920 X 2
<i>filterWindow327680ns</i>	2	81920 X 4
<i>filterWindow655360ns</i>	3	81920 X 8
<i>filterWindow1310720ns</i>	4	81920 X 16
<i>filterWindow2621440ns</i>	5	81920 X 32
<i>filterWindow5242880ns</i>	6	81920 X 64
<i>filterWindow10485760ns</i>	7	81920 X 128
<i>filterWindow20971520ns</i>	8	81920 X 256
<i>filterWindow41943040ns</i>	9	81920 X 512
<i>filterWindow83886080ns</i>	10	81920 X 1024
<i>filterWindow167772160ns</i>	11	81920 X 2048
<i>filterWindow335544320ns</i>	12	81920 X 4096
<i>filterWindow671088640ns</i>	13	81920 X 8192
<i>filterWindow1342177280ns</i>	14	81920 X 16384
<i>filterWindow2684354560ns</i>	15	81920 X 32768

Option	Value	Usage
filterWindow5368709120ns	16	81920 X 65536

COMMANDS

The *packetGroupThresholdList* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

packetGroupThresholdList clear option

This clears threshold timestamps. The threshold value list is empty.

packetGroupThresholdList config option

Allows setting the values (options).

Example: *packetGroupThresholdList config threshold 250000*

packetGroupThresholdList getFirst

Access the first value in the list. Specific errors are:

- There are no values in the list

packetGroupThresholdList getNext

Access the next value in the list. Specific errors are:

- There are no more values in the list

packetGroupThresholdList add

Add a value (a 'from' and a 'to') to the threshold value list.

packetGroupThresholdList setDefault

Sets to IxTclHal default values for all configuration options. Does not clear the list.

EXAMPLES

Example TCL commands to turn Jitter Filter on and configure threshold and mask.

```

package req IxTclHal

ixConnectToChassis <chassis>

set chassisId 1
set cardId 3
set portId 2

portCpu reset $chassisId $cardId $portId

port setFactoryDefaults $chassisId $cardId $portId

set receiveMode [expr $::portRxModeWidePacketGroup |
$::portRxModeRateMonitoring]

port setReceiveMode $receiveMode $chassisId $cardId $portId

packetGroup getRx $chassisId $cardId $portId

packetGroupThresholdList setDefault
packetGroupThresholdList config -enableJitterFilter true
packetGroupThresholdList config -fromPGID 0
packetGroupThresholdList config -toPGID 2

```

```
packetGroupThresholdList config -threshold 1000
packetGroupThresholdList config -filterWindow filterWindow81920ns

packetGroupThresholdList add

packetGroup setRx $chassisId $cardId $portId

port write $chassisId $cardId $portId
```

SEE ALSO*packetGroup*

NAME - packetLengthInsertion

packetLengthInsertion - used to insert packet length value. The packet length that is inserted is the hex form of the difference between the actual packet length and the adjustment value that will be specified.

SYNOPSIS	packetLengthInsertion <i>sub-command options</i>
-----------------	--

DESCRIPTION	The <i>packetLengthInsertion</i> command is used to insert packet length value. The packet length that is inserted is the hex form of the difference between the actual packet length and the adjustment value that will be specified.
--------------------	--

STANDARD OPTIONS	
-------------------------	--

enabled	enables the insertion of packet length option.
offset	the offset at which the packet length needs to be inserted in a packet.
adjustment	number of bytes that needs to be adjusted for the packet length.

COMMANDS	The <i>packetLengthInsertion</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

packetLengthInsertion **set** *option*

sets the packet length insertion option.

packetLengthInsertion **get** *option*

gets the packet length insertion option.

packetLengthInsertion **del**

deletes the packet length insertion option.

packetLengthInsertion **add**

adds the packet length insertion option.

packetLengthInsertion **getCount**

gets the count of packet length.

EXAMPLES	The following example shows how to insert the difference in the length of the packet and adjustment value of 'abc' in the stream at offset 'xyz' bytes in the packet. This is set to random frame size to see the change in values.
-----------------	---

```
package req IxTclHal
ixConnectToChassis xm12-qa7

port get 1 11 1
port config -loopback true
port set 1 11 1
port write 1 11 1
```

```

stream get 1 11 1 1
stream config -frameSizeType sizeRandom
stream config -frameSizeMIN 64
stream config -frameSizeMAX 100
stream config -dma                                     stopStream

stream set 1 11 1 1
stream write 1 11 1 1

stream get 1 11 1 1

packetLengthInsertion add 3
packetLengthInsertion getCount

packetLengthInsertion get 1
packetLengthInsertion config -enabled true
packetLengthInsertion config -offset 16
packetLengthInsertion config -adjustment 4
packetLengthInsertion set 1

packetLengthInsertion get 2
packetLengthInsertion config -enabled 2
packetLengthInsertion config -offset 18
packetLengthInsertion config -adjustment 8
packetLengthInsertion set 2

packetLengthInsertion get 3
packetLengthInsertion config -enabled 3
packetLengthInsertion config -offset 20
packetLengthInsertion config -adjustment 16
packetLengthInsertion set 3

stream set 1 11 1 1
stream write 1 11 1 1

set offset1 16 ; set adjustment1 4
set offset2 20 ; set adjustment2 8
set offset3 24 ; set adjustment3 16

stream get 1 11 1 1
stream getPacketView 1
set packetView [stream cget -packetView]

after 5000
set prtList [list [list 1 11 1]]
ixClearStats prtList
ixStartCapture prtList
ixStartPortTransmit 1 11 1

after 5000
ixStopCapture prtList

after 5000
captureBuffer get 1 11 1
captureBuffer getframe 1
set captureFrame [captureBuffer cget -frame]

set totalLength1 [llength $packetView]
set totalLength2 [llength $captureFrame]

```

```
foreach offset "$offset1 $offset2 $offset3" adjustment
"$adjustment1 $adjustment2 $adjustment3" {
foreach frame "[list $packetView] [list $captureFrame]" length
"$totalLength1 $totalLength2" {
puts "OFFSET $offset ADJUSTMENT $adjustment LENGTH $length"

set hexLength($offset) [join [lrange $frame $offset [expr $offset
+ 1] ] ""]
puts "HEX $hexLength($offset)"
set decLength($offset) [expr 0x$hexLength($offset)]
puts "DEC $decLength($offset)"

if { $decLength($offset) == [expr $length - $adjustment] } {
puts "PASS for offset $offset"
} else {
puts "FAIL for offset $offset"
}
}
}
```

NAME - pauseControl

pauseControl - configure a pause control packet.

SYNOPSIS	pauseControl <i>sub-command options</i>										
DESCRIPTION	The pauseControl command is used to configure the parameters on a stream to transmit pause control frames.										
STANDARD OPTIONS											
da	(Read-only, except for 10GE cards) The MAC address of the interface receiving the pause control message. (<i>default</i> = 01 80 C2 00 00 01)										
pauseControlType	Use to configure the priority control type. (<i>default</i> = ieee8023x)										
<table border="1"> <thead> <tr> <th>Option</th><th>Value</th><th>Usage</th></tr> </thead> <tbody> <tr> <td>ieee8023x</td><td>0</td><td>(<i>default</i>) IEEE 802.3x values: The Length/Type for a MAC Control frame = 88 08. The MAC Control Opcode for the PAUSE control function = 00 01. Pause Quanta = 255: The user-specified pause counter value, measured in Pause Quanta units. (1 Pause Quanta = 512 bit times.) Valid range is 0 to 65535 pause quanta.</td></tr> <tr> <td>ieee8021Qbb</td><td>1</td><td>(<i>default</i>) IEEE 802.1Qbb values: The Length/Type for a MAC Control frame = 88 08. The MAC Control Opcode for the PAUSE control function = 01 01. Priority Enable Vector = 00 00 Pause Quanta = '00...00' (16 octets) where each pair contains the enable/disable value and pause quanta value.</td></tr> </tbody> </table>			Option	Value	Usage	ieee8023x	0	(<i>default</i>) IEEE 802.3x values: The Length/Type for a MAC Control frame = 88 08. The MAC Control Opcode for the PAUSE control function = 00 01. Pause Quanta = 255: The user-specified pause counter value, measured in Pause Quanta units. (1 Pause Quanta = 512 bit times.) Valid range is 0 to 65535 pause quanta.	ieee8021Qbb	1	(<i>default</i>) IEEE 802.1Qbb values: The Length/Type for a MAC Control frame = 88 08. The MAC Control Opcode for the PAUSE control function = 01 01. Priority Enable Vector = 00 00 Pause Quanta = '00...00' (16 octets) where each pair contains the enable/disable value and pause quanta value.
Option	Value	Usage									
ieee8023x	0	(<i>default</i>) IEEE 802.3x values: The Length/Type for a MAC Control frame = 88 08. The MAC Control Opcode for the PAUSE control function = 00 01. Pause Quanta = 255: The user-specified pause counter value, measured in Pause Quanta units. (1 Pause Quanta = 512 bit times.) Valid range is 0 to 65535 pause quanta.									
ieee8021Qbb	1	(<i>default</i>) IEEE 802.1Qbb values: The Length/Type for a MAC Control frame = 88 08. The MAC Control Opcode for the PAUSE control function = 01 01. Priority Enable Vector = 00 00 Pause Quanta = '00...00' (16 octets) where each pair contains the enable/disable value and pause quanta value.									
pauseFrame	Use to configure the hex byte priorities; 16 byte hex list. (<i>default</i> = '00....00')										
pauseTime	The pause time, measured in pause quanta units. (1 Pause Quanta = 512 bit times.) The valid range is 0 to 65535 pause quanta. (<i>default</i> = 255)										
pfcEnableValueList	Use to configure the priority parameters using pair list, where each pair contains the enable/disable value and pause quanta value. Only used when <i>usePfcEnableValueList</i> = true. (<i>default</i> = '{0 0} {0 0} {0 0} {0 0} {0 0} {0 0} {0 0} {0 0}')										
priorityEnableVector	Use to configure the priority enable vector. (<i>default</i> = '00 00')										
usePfcEnableValueList <i>true/false</i>	Use to choose between <i>pauseFrame</i> or <i>pfcEnableValueList</i> . (<i>default</i> = false)										

COMMANDS

The *pauseControl* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

pauseControl cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *pauseControl* command.

pauseControl config *option value*

Modify the configuration options of the *pauseControl*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *pauseControl*.

pauseControl decode *capSlice [chassisID cardID portID]*

Decodes a captured slice/frame into the pause control variables. If not an pause control frame, returns TCL_ERROR. May be used to determine if the captured frame is a valid pause control frame. Specific errors are:

- No connection to a chassis
- The captured frame is not an pause control frame

pauseControl get *chassisID cardID portID*

Gets the current configuration of the *pauseControl* frame for port with id *portID* on card *cardID*, chassis *chassisID*. from its hardware. Call this command before calling *pauseControl cget option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

pauseControl set *chassisID cardID portID*

Sets the configuration of the pause control frame in IxHAL for port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the *pauseControl config option value* command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

pauseControl setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package req IxTclHal

set hostname loopback

if {[ixConnectToChassis $hostname]} {
    errorMsg "error connecting $hostname chassis"
    return "FAIL"
}

set chassis      [chassis cget -id]
set card         2
set port         1
set streamId1
set portList [list [list $chassis $card $port] ]
```

```

port setFactoryDefaults $chassis $card $port
port config -enableDataCenterMode true
port config -flowControlType ieee8021Qbb
port config -pfcEnableValueList "{1 0} {0 2} {1 1}
{0 2} {1 3} {0 3} {0 3} {1 1}"

if {[port set $chassis $card $port]} {
    errorMsg "Error calling port set $chassis $card $port"
    set retCode $::TCL_ERROR
}

ixWritePortsToHardware portList
ixCheckLinkState portList

stream setDefault
stream config -name "PFC stream"
stream config -priorityGroup priorityGroup0

stream setDefault

protocol setDefault
protocol config -name pauseControl
protocol config -ethernetType ethernetII

pauseControl     setDefault
pauseControl      config      -da           {01 80 C2 00
00 01}
pauseControl      config      -pauseTime     128
if {[pauseControl set $chassis $card $port]} {
    errorMsg "Error calling pauseControl set $chassis $card
$port"
    set retCode $::TCL_ERROR
}

if {[stream set $chassis $card $port $streamId]} {
    errorMsg "Error calling stream set $chassis $card $port
$streamId"
    set retCode $::TCL_ERROR
}

# second stream
incr streamId
protocol setDefault
protocol config -name pauseControl
protocol config -ethernetType ethernetII
pauseControl setDefault
pauseControl config -pauseControlType ieee8021Qbb
pauseControl config -usePfcEnableValueList $::true
pauseControl config -pfcEnableValueList "{1 555} {0 0}
{0 0} {0 0} {1 2} {1 6} {0 0} {0 0}"
if {[pauseControl set $chassis $card $port]} {
    errorMsg "Error calling pauseControl set $chassis $card
$port"
    set retCode $::TCL_ERROR
}

if {[stream set $chassis $card $port $streamId]} {
    errorMsg "Error calling stream set $chassis $card $port
$streamId"
    set retCode $::TCL_ERROR
}

```

A**pauseControl**

```
ixWriteConfigToHardware portList -noProtocolServer
```

SEE ALSO*ip*

NAME - pcsLaneError

pcsLaneError - configure PCS lane errors.

SYNOPSIS	pcsLaneError <i>sub-command options</i>									
DESCRIPTION	The pcsLaneError command is used to insert errors into PCS lanes, either only the Lane Marker fields or into both Lane Markers and Payload fields.									
<hr/>										
STANDARD OPTIONS										
periodType	Use to configure the PCS Error Period Type. (<i>default = pcsLaneErrorPeriodTypeLaneMarkers</i>)									
	<table border="1"> <thead> <tr> <th>Option</th><th>Value</th><th>Usage</th></tr> </thead> <tbody> <tr> <td>pcsLaneErrorPeriodTypeLaneMarkers</td><td>0</td><td>Lane Markers period type (only)</td></tr> <tr> <td>pcsLaneErrorPeriodTypeLaneMarkersAndPayload</td><td>1</td><td>both Lane Markers and Payload period types</td></tr> </tbody> </table>	Option	Value	Usage	pcsLaneErrorPeriodTypeLaneMarkers	0	Lane Markers period type (only)	pcsLaneErrorPeriodTypeLaneMarkersAndPayload	1	both Lane Markers and Payload period types
Option	Value	Usage								
pcsLaneErrorPeriodTypeLaneMarkers	0	Lane Markers period type (only)								
pcsLaneErrorPeriodTypeLaneMarkersAndPayload	1	both Lane Markers and Payload period types								
enableContinuous <i>true/false</i>	If set to <i>true</i> , transmits errors continuously at the given period and count. If <i>false</i> , see <i>repeat</i> , below. (<i>default = false</i>)									
pcsLane	Specifies which lane to insert errors into. Valid values range 0–19 for 100G load modules; 0–3 for 40G load modules. (<i>default = 0</i>)									
period	Periodicity of transmitted errors. The unit of period differs based on the type of error (periodType) selected. (<i>default = 1</i>) <ul style="list-style-type: none"> • Type = lane markers, period = lane markers • Type = lane markers and payload, period = 64/66 bit words 									
count	Consecutive errors to transmit (<i>default = 1</i>)									
repeat	Total number of errors to transmit. This is value ignored if <i>enableContinuous</i> is set to true. (<i>default = 1</i>)									
syncBits	Hex field for entering the error bits for the sync field (<i>default = 0x00</i>)									
laneMarkerFields	Hex field for entering the lane marker fields (<i>default = 00 00 00 00 00 00 00 01</i>)									
configuredErrorBits	(Read-only) Resultant configuration of bits to be sent out on the pcs lane, including the two sync bits. Result returned in string binary format similar to IxExplorer for all 8 bytes + 2 sync bits.									

COMMANDS	The <i>pcsLaneError</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	---

pcsLaneError get *chasID cardID portID*

Gets the current configuration of the PCS lane error from IxHAL for port with id *portID* on card *cardID*, chassis *chasID*. Specific errors are:

A

pcsLaneError

- No connection to a chassis
- Invalid port number

pcsLaneError set *chasID cardID portID*

Sets the configuration of the PCS lane error in IxHAL for port with id *portID* on card *cardID*, chassis *chasID*. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

pcsLaneError start *chasID cardID portID*

Starts the transmission of PCS lane errors for port with id *portID* on card *cardID*, chassis *chasID*. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

pcsLaneError stop *chasID cardID portID*

Stops the transmission of PCS lane errors for port with id *portID* on card *cardID*, chassis *chasID*. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

pcsLaneError setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See [pcsLaneStatistics](#)

SEE ALSO

[pcsLaneStatistics](#), [txLane](#)

NAME - pcsLaneStatistics

pcsLaneStatistics - retrieves the rx stats per each physical lane.

SYNOPSIS

pcsLaneStatistics *sub-command options*

DESCRIPTION

The **pcsLaneStatistics** command is used to retrieve the rx stats per each physical lane.

STANDARD OPTIONS

syncHeaderLock

Indicates if the received PCS lane achieved sync-bit lock. (*default = pcsLaneStateLock*)

Option	Value	Usage
pcsLaneStateLock	1	lane state: lock
pcsLaneStateNoLock	0	lane state: not locked

pcsLaneMarkerLock

Indicates if the received PCS lane has achieved alignment marker lock. (*default = pcsLaneStateLock*)

Option	Value	Usage
kError	0	red led, currently there is an error condition
kNa	1	green led, not used in lostPcsLaneMarkerLock
k.NoError	2	grey led, no error condition since clearing stats
kLatched	3	yellow led, there was an error since the last clear but no error at present

pcsLaneMarkerMap

The PCS lane number identified by the alignment marker.

relativeLaneSkew

Shows the actual skew in nanoseconds. Skew measurements are valid only when all lanes are locked with 20 unique lane markers. The first lane markers to arrive have skew of 0. All other lane skews are relative to them.

syncHeaderErrorCount

The number of synchronization bit errors received.

pcsLaneMarkerErrorCount

The number of incorrect PCS lane markers received while in PCS lane lock state.

bip8ErrorCount

The number of BIP-8 errors for a PCS lane. BIP-8 = Bit-Interleaved Parity with eight bit errors (BIP-8). Each bit in the BIP field is an even parity calculation over all previous selected bits of a PCS lane.

lostSyncHeaderLock

Indicates the loss of sync header lock since the last statistic was read. (*default = pcsLaneStateLock*)

Option	Value	Usage
pcsLaneStateLock	1	lane state: lock

Option	Value	Usage
pcsLaneStateNoLock	0	lane state: not locked

lostPcsLaneMarkerLock Indicates the loss of PCS lane marker lock since the last statistic was read.
(*default = pcsLaneStateLock*)

Option	Value	Usage
kError	0	red led, currently there is an error condition
kNa	1	green led, not used in lostPcsLaneMarkerLock
k.NoError	2	grey led, no error condition since clearing stats
kLatched	3	yellow led, there was an error since the last clear but no error at present

COMMANDS

The *pcsLaneStatistics* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

pcsLaneStatistics get *chassisID cardID portID*

Retrieves the rx PCS lane skew statistics from ixServer for port with id *portID* on card *cardID*, chassis *chassisID*. Specific errors are:

- No connection to a chassis
- Invalid port number

pcsLaneStatistics getLane *physicalLane*

Retrieves the data from local IxHAL after the *get* has been issued and updates the object with one row of the rx lane stats, indexed per physical lane id. Specific errors are:

- No connection to a chassis
- Invalid physical lane

pcsLaneStatistics setDefault

Sets to IxTclHal default values for all configuration options.

clearPcsLaneStatistics *portList*

This command eliminates all pcs data lane stats on all ports specified in *portList*.

EXAMPLES

```
package req IxTclHal

set hostnameloopback

if {[ixConnectToChassis $hostname]} {
    errorMsg "error connecting $hostname chassis"
    return "FAIL"
}
set chasId      [chassis cget -id]
set cardId      140
set portId      1

set portList [list [list $chasId $cardId $portId ] ]
```


ixPuts
cleanUp

SEE ALSO

pcsLaneError, txLane

NAME - pcpuCommandService

pcpuCommandService - execute Linux commands on a port's CPU

SYNOPSIS	pcpuCommandService <i>sub-command options</i>
DESCRIPTION	<p>Most intelligent Ixia ports runs the Linux Operating system. Any Linux command may be remotely executed by TCL programming. The <i>port</i> command's <i>isValidFeature</i> sub-command may be used to determine if a given port runs Linux. Use the following sequence:</p> <pre>if [port isValidFeature \$chassis \$card \$port portFeatureIxRouter] { ... port runs Linux ... }</pre> <p>Refer to Issue Port CPU Command on page 5-123 for an overview of this command. Commands may be sent to a set of ports and executed simultaneously. Different commands may be executed on different ports. The result of each port's command execution may be individually retrieved.</p> <p>The <i>add</i> sub-command is used to build a list of commands for multiple ports. The <i>execute</i> command causes all commands in the list to be sent to the affected ports and executed simultaneously. The result of all command execution is available by traversing through the list using the <i>getFirst</i> and <i>getNext</i> sub-commands. All <i>Standard Options</i> are read-only and only valid after a <i>getFirst/getNext</i> call.</p>

STANDARD OPTIONS

cardID	<i>Read-only</i> . The card associated with the command.
chassisID	<i>Read-only</i> . The chassis associated with the command.
command	<i>Read-only</i> . The executed command.
error	<i>Read-only</i> . After command execution, the first 1024 characters that were sent to the standard error stream.
input	<i>Read-only</i> . Optional text to be used as the standard input stream for the <i>command</i> to be executed.
output	<i>Read-only</i> . After command execution, the first 1024 characters that were sent to the standard output stream.
portID	<i>Read-only</i> . The port associated with the command.
result	<i>Read-only</i> . After command execution, the return code from the command. Normally, '0' indicates a successful command execution and non-zero indicates an error.

COMMANDS

The **pcpuCommandService** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

pcpuCommandService add *chassisID cardID portID command [input]*

Adds a command to the indicated port. The same command may be entered multiple times; commands are executed in the order that the *add* sub-command was used.

command is the text of the command to be executed, which must use an absolute path. For example, ‘/bin/ls’. No filename expansion is performed on the command. For example, ‘/bin/ls /bin/ix*’ finds no matches. This, and the restriction on absolute path, may be avoided by executing the command through a *bash* shell, as in:

```
pcpuCommandService add 1 1 1 "/bin/bash -c 'ls -l /bin/ix*'"
```

The *input* argument is optional, and if present, is used as the standard input stream for the *command*. For example, the following echos ‘hello world’ to the commands standard output stream.

```
set command "/bin/cat"
set input "hello world\n"
pcpuCommandService add 1 1 1 $command $input
```

Specific errors are:

- No connection to a chassis
- Invalid port specification
- The port is owned by another user
- The port does not support Linux

pcpuCommandService cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *pcpuCommandService* command.

pcpuCommandService del *chassisID cardID portID*

Removes all commands for the indicated port. Specific errors are:

- No connection to a chassis
- Invalid port specification
- The port is owned by another user
- The port does not support Linux

pcpuCommandService execute

All commands for all ports are sent to the ports for execution. The results of the commands’ execution is available in the *Standard Options* after calls to *getFirst* and *getNext*. Specific errors are:

- Communications error with one or more ports

pcpuCommandService getFirst

Access the first command in the list. The command’s results are available in the standard options. Specific errors are:

- There are no commands in the list

pcpuCommandService **getNext**

Access the next command in the list. The command's results are available in the standard options. Specific errors are:

- There are no more commands in the list

pcpuCommandService **setDefault**

Sets default values for all configuration options and clears all commands from the list.

EXAMPLES

```
package require IxTclHal

set host localhost
set username user
# Assume card 1 is a card that supports Linux
set card 1

# Commands to execute on ports
# Odd ports will echo a command from standard input
set oddCmd "/bin/cat"
set oddInput "hello there\n"

# Even ports will execute a command through bash
# This allows PATH lookup and filename expansion
set evenCmd "/bin/bash -c 'ls -l /bin/ix*'

# If this is a UNIX system, connect through TCL Server
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID
set chas [ixGetChassisID $host]

# Make sure that this card supports Linux
if {[port isValidFeature $chas $card 1 portFeatureIxRouter] == 0}
{
    ixPuts "$chas:$card does not have a local CPU"
    return 1
}

# Get the number of ports on the card
if [card get $chas $card] {
    ixPuts $::ixErrorInfo
    return 1
}
set portCount [card cget -portCount]

# Delete any previous list of commands
pcpuCommandService setDefault
```

```
# For all the ports
for {set i 1} {$i <= $portCount} {incr i 1} {
    # For the odd ports
    if {[expr $i & 1] } {
        if [pcpuCommandService add $chassis $card $i $oddCmd \
            $oddInput] {
            ixPuts ::ixErrorInfo
            return 1
        }
    } else {
        # for the even ports
        if [pcpuCommandService add $chassis $card $i $evenCmd ] {
            ixPuts ::ixErrorInfo
            return 1
        }
    }
}

# Do the commands
if [pcpuCommandService execute] {
    ixPuts ::ixErrorInfo
    return 1
}

# Retreive and print the results
for {set next [pcpuCommandService getFirst]} \
    {$next != ::TCL_ERROR} \
    {set next [pcpuCommandService getNext]} {
    set chassis [pcpuCommandService cget -chassisID]
    set card [pcpuCommandService cget -cardID]
    set port [pcpuCommandService cget -portID]
    set command [pcpuCommandService cget -command]
    set output [pcpuCommandService cget -output]
    set result [pcpuCommandService cget -result]

    ixPuts -none newline "$chassis:$card:$port, "
    ixPuts "cmd: $command, result: $result, output: $output"
}
```

SEE ALSO

port

NAME - poeAutoCalibration

poeAutoCalibration - initiate and query PoE port calibration

SYNOPSIS

poePoweredDevice *sub-command options*

DESCRIPTION

The *poePoweredDevice* command is used to initiate a PoE port calibration and/or determine the status of a calibration. Calibration of all PoE ports is performed at chassis power-up time.

A calibration is initiated by calling the *initiateCalibrate* sub-command. The calibration may take up to 20 seconds. The results of a calibration, either while it is proceeding or after it has completed, can be determined by first calling *requestStatus*, waiting a second and then calling *get*. The status of the calibration is then available through the options in this command.

STANDARD OPTIONS

currentReadbackStatus *Read-only*. The status of the calibration procedure for current readback.

Option	Value	Usage
poeAutoCalibrationTesting	0	Calibration is still in progress.
poeAutoCalibrationPass	1	The calibration completed successfully.
poeAutoCalibrationFail	2	The calibration failed.

iClassRangeStatus *Read-only*. The status of the calibration procedure for the class range. See *currentReadbackStatus* for the possible values of this option.

iLoadRangeStatus *Read-only*. The status of the calibration procedure for the load range. See *currentReadbackStatus* for the possible values of this option.

iPulseRangeStatus *Read-only*. The status of the calibration procedure for the pulse range. See *currentReadbackStatus* for the possible values of this option.

voltageReadbackStatus *Read-only*. The status of the calibration procedure for voltage readback. See *currentReadbackStatus* for the possible values of this option.

COMMANDS

The *poePoweredDevice* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

poeAutoCalibration cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *poeAutoCalibration* command.

poeAutoCalibration get *chassisID cardID portID*

Gets the current configuration of the *poeAutoCalibration* parameters on the indicated port from its hardware. Call this command before calling *poeAutoCalibration cget *option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis

- Invalid port number

poeAutoCalibration **initiateCalibrate *chasID cardID portID***

Initiates the calibration process on the indicated port. The end of the calibration process may be determined by calling *requestStatus* and *get* or by waiting 20 seconds. Specific errors are:

- No connection to a chassis
- Invalid port number

poeAutoCalibration **requestStatus *chasID cardID portID***

Requests that the status of the calibration be retrieved from the port indicated. The values may take up to a second to be read back. A call to this sub-command should be followed by a call to *get*. Specific errors are:

- No connection to a chassis
- Invalid port number

poeAutoCalibration **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *poePoweredDevice*.

SEE ALSO

poePoweredDevice

NAME - poePoweredDevice

poePoweredDevice - control Power over Ethernet Powered Device (PD) emulation

SYNOPSIS

poePoweredDevice *sub-command options*

DESCRIPTION

The *poePoweredDevice* command is used to setup the parameters by which a PoE Powered Device (PD) is emulated on a port.

The port can emulate a device that uses either *Alternative A* and/or *Alternative B*. This is controlled by the *relayControl* option.

The emulated class is controlled by the *enableClassSignature* and *signatureValue* options; the *classType* indicates the calculated class based on the signature value.

The emulated detection signature is controlled by the *enableDetectionSignature*, *rsig*, *csig* and *enableCsig10uF* options.

The emulated Alternating Current Maintain Power Signature (ACMPS) is controlled by the *enableAcMpsSignature*, *rpd*, *cpd* and *enable CpdAdd10uF* options.

Once the PSE (Power Sourcing Equipment) has classified the emulated PD, it should provide power to the port. The power requirements of the emulated port are controlled by the *steadyStateLoadControl*, *controlledCurrent*, *controlledPower* and *idleCurrent* options. Transient load variations may be inserted through the use of the *enableTransientLoadControl*, *transientLoadControl*, *pulseWidth*, *duty*, *pulsedCurrent* and *slewRate* options. Pulses are applied through the use of the *portGroup setCommand* sub-command, with an *loadPoEPulse* value or through the high-level *ixLoadPoePulse* and *ixLoadPortPoePulse* commands; if *enableTransientLoadControl* is true and *transientLoadControl* is set to *poeLoadControlSinglePusle*, then a pulsed current as indicated by *pulsedCurrent* and *slewRate* is injected for the period indicated by *pulseWidth*.

The voltage thresholds that are used by the PD to detect state transitions may be set by the *vOperate*, *vOff*, *vClassify*, *vDetect* and *vNoop* options.

STANDARD OPTIONS

classType

Read-only. If *enableClassSignature* is true, this is the calculated classification of *signatureValue*.

Option	Value	Usage
poeClass0	0	Class 0.
poeClass1	1	Class 1.
poeClass2	2	Class 2.
poeClass3	3	Class 3.
poeClass4	4	Class 4.
poeMaybeClass0or1	5	Either class 0 or 1.
poeMaybeClass0or1or2	6	Either class 0, 1 or 2.

Option	Value	Usage
poeMaybeClass0or2or3	7	Either class 0, 2 or 3.
poeMaybeClass0or3or4	8	Either class 0, 3 or 4.
poeMaybeClass0or4	9	Either class 0 or 4.
poeClassTypeUndefined	10	Unknown classification.

controlledCurrent

If *steadyStateLoadControl* is set to *poeLoadControlConstantCurrent*, then this is the amount of current that the PD requires from the PSE, in mA. The value may be between 0 and 600mA. (*default* = 42.5)

controlledPower

If *steadyStateLoadControl* is set to *poeLoadControlConstantPower*, then this is the amount of power that the PD requires from the PSE, in watts. The value may be between 0 and 20W. (*default* = 2.0)

cpd

If *enableAcMpsSignature* is true, this is the capacitance signature expressed in nFarads, between 0 and 220nF. If *enableCpdAdd10uF* is set, then 10uF of capacitance is added to this value, effectively overriding it. (*default* = 50)

csig

If *enableDetectionSignature* is true, this is the capacitance signature expressed in nFarads, between 0 and 220nF. If *enableCsigAdd10uF* is set, then 10uF of capacitance is added to this value, effectively overriding it. (*default* = 50)

duty

If *enableTransientLoadControl* is true and *transientLoadControl* is set to *poeLoadControlContinuousPulse*, then this is the duty cycle of the transient load. This is expressed as a percentage of total time that transient loads is injected. (*default* = 30)

**enableAcMpsSignature
true / false**

If true, then the ACMPS signature is set from the values in *rpd*, *cpd* and *enableCpdAdd10uF*. (*default* = true)

**enableClassSignature
true / false**

If true, then the *signatureValue* option is used to set the emulated class. The computed class name is indicated in *classType*. (*default* = true)

**enableCpdAdd10uF
true / false**

If *enableAcMpsSignature* is true, then if this option is true, a value of 10uF is added to the *cpd* value which sets the capacitance signature. This effectively overrides the *cpd* value, which is expressed in nF. (*default* = false)

**enableCsigAdd10uF
true / false**

If *enableDetectionSignature* is true, then if this option is true, a value of 10uF is added to the *csig* value which sets the capacitance signature. This effectively overrides the *csig* value, which is expressed in nF. (*default* = false)

**enableDetection
Signature true / false**

Enables the use of the *rsig*, *csig* and *enableCsigAdd10uF* options to set the PoE detection signature. (*default* = true)

**enablePulseOnStart
true / false**

If true, then a single pulse is sent each time that the PSE starts to apply power. (*default* = false)

**enableTransientLoad
Control true / false**

If true, then transient loads is injected based on the values in the *transientLoadControl*, *pulseWidth*, *duty*, *pulsedCurrent* and *slewRate* options. (*default* = true)

idleCurrent	If <i>steadyStateLoadControl</i> is set to <i>poeLoadControlIdle</i> , then this is the amount of current that the PD requires from the PSE, in mA. The value may be between 0 and 16mA. (<i>default</i> = 10)															
pulseWidth	If <i>enableTransientLoadControl</i> is <i>true</i> and <i>transientLoadControl</i> is set to <i>poeLoadControlSinglePusle</i> , then this is the width of the transient pulse that is injected. This is expressed in msec. (<i>default</i> = 40.25)															
pulsedCurrent	If <i>enableTransientLoadControl</i> is <i>true</i> and <i>transientLoadControl</i> is set to <i>poeLoadControlContinuousPusle</i> , then this is the current injected. This is expressed in mA and may be less than or greater than the steady state value. (<i>default</i> = 333.0)															
relayControl	This option controls the combination of power options are supported by the PD.															
	<table border="1"> <thead> <tr> <th>Option</th><th>Value</th><th>Usage</th></tr> </thead> <tbody> <tr> <td>poeRelayControlNoMode</td><td>0</td><td>Neither mode is supported.</td></tr> <tr> <td>poeRelayControlAlternativeA</td><td>1</td><td>Use Alternative A.</td></tr> <tr> <td>poeRelayControlAlternativeB</td><td>2</td><td>Use Alternative B.</td></tr> <tr> <td>poeRelayControlBothAandB</td><td>3</td><td>(<i>default</i>) Use Alternative A or B.</td></tr> </tbody> </table>	Option	Value	Usage	poeRelayControlNoMode	0	Neither mode is supported.	poeRelayControlAlternativeA	1	Use Alternative A.	poeRelayControlAlternativeB	2	Use Alternative B.	poeRelayControlBothAandB	3	(<i>default</i>) Use Alternative A or B.
Option	Value	Usage														
poeRelayControlNoMode	0	Neither mode is supported.														
poeRelayControlAlternativeA	1	Use Alternative A.														
poeRelayControlAlternativeB	2	Use Alternative B.														
poeRelayControlBothAandB	3	(<i>default</i>) Use Alternative A or B.														
rpd	If <i>enableAcMpsSignature</i> is true, this is the ACMPS resistance signature expressed as a floating point value between 10 and 40 kOhms. (<i>default</i> = 23.0)															
rpdRangeControl	On newer, 30watt PoE modules, it is possible to change the range associated with <i>rpd</i> .															
	<table border="1"> <thead> <tr> <th>Option</th><th>Value</th><th>Usage</th></tr> </thead> <tbody> <tr> <td>poeRpdRangeZac1</td><td>0</td><td>(<i>default</i>) The range of <i>rpd</i> is from 10 - 45kOhm.</td></tr> <tr> <td>poeRpdRangeZac2</td><td>1</td><td>The range of <i>rpd</i> is from 200 - 1200kOhm.</td></tr> </tbody> </table>	Option	Value	Usage	poeRpdRangeZac1	0	(<i>default</i>) The range of <i>rpd</i> is from 10 - 45kOhm.	poeRpdRangeZac2	1	The range of <i>rpd</i> is from 200 - 1200kOhm.						
Option	Value	Usage														
poeRpdRangeZac1	0	(<i>default</i>) The range of <i>rpd</i> is from 10 - 45kOhm.														
poeRpdRangeZac2	1	The range of <i>rpd</i> is from 200 - 1200kOhm.														
rsig	If <i>enableDetectionSignature</i> is true, this is the resistance signature expressed as a floating point value between 10 and 40 kOhms. (<i>default</i> = 17.0)															
signatureValue	If <i>enableClassSignature</i> is true, the class signature value, expressed as a floating point number between 0mA and 60mA. (<i>default</i> = 18.5)															
slewRate	If <i>enableTransientLoadControl</i> is <i>true</i> and <i>transientLoadControl</i> is set to <i>poeLoadControlContinuousPusle</i> , then this is the slew rate at which the current indicated in <i>pulsedCurrent</i> is injected. This is expressed in mA/msec. (<i>default</i> = 33.0)															
steadyStateLoad Control	This option controls the type of power requirements for the emulated PD after classification has completed.															
	<table border="1"> <thead> <tr> <th>Option</th><th>Value</th><th>Usage</th></tr> </thead> <tbody> <tr> <td>poeLoadControlConstantCurrent</td><td>0</td><td>(<i>default</i>) The PD requires constant current, as indicated in the <i>controlledCurrent</i> option.</td></tr> <tr> <td>poeLoadControlControlledPower</td><td>1</td><td>The PD requires controlled power, as indicated in the <i>controlledPower</i> option.</td></tr> <tr> <td>poeLoadControlIdle</td><td>2</td><td>The PD requires constant current, as indicated in the <i>idleCurrent</i> option.</td></tr> </tbody> </table>	Option	Value	Usage	poeLoadControlConstantCurrent	0	(<i>default</i>) The PD requires constant current, as indicated in the <i>controlledCurrent</i> option.	poeLoadControlControlledPower	1	The PD requires controlled power, as indicated in the <i>controlledPower</i> option.	poeLoadControlIdle	2	The PD requires constant current, as indicated in the <i>idleCurrent</i> option.			
Option	Value	Usage														
poeLoadControlConstantCurrent	0	(<i>default</i>) The PD requires constant current, as indicated in the <i>controlledCurrent</i> option.														
poeLoadControlControlledPower	1	The PD requires controlled power, as indicated in the <i>controlledPower</i> option.														
poeLoadControlIdle	2	The PD requires constant current, as indicated in the <i>idleCurrent</i> option.														

Option	Value	Usage
poeLoadControlShutdown	3	The PD is in shutdown mode.

transientLoadControl

If *enableTransientLoadControl* is true, then this option indicates the type of transient load that is injected.

Option	Value	Usage
poeLoadControlSinglePulse	0	Inject a transient load once when the <i>pulse</i> sub-command is used.
poeLoadControlContinuousPulse	1	(default) Inject transient loads continuously.

vClassify

The maximum voltage for the emulated PD classification stage. Between this setting and *vDetect*, the classification currents are presented to the PSE by the PD. (default = 20.5)

vDetect

The maximum voltage for emulated PD detection. Between this setting and *vNoop*, the detection signature impedances are presented to the PSE by the PD. (default = 10.0)

vNoop

The minimum detection voltage. No signatures are presented below this threshold value. (default = 2.8)

vOff

Sets the input threshold below which the PSE load is removed. (default = 33.0)

vOperate

Sets the input threshold where the PSE load is first applied. (default = 38.0)

COMMANDS

The *poePoweredDevice* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

poePoweredDevice cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *poePoweredDevice* command.

poePoweredDevice config *option value*

Modify the *poePoweredDevice* configuration options of the port. If no *option* is specified, returns a list describing all of the available *poePoweredDevice* options (see STANDARD OPTIONS) for port.

poePoweredDevice get *chassisID cardID portID*

Gets the current configuration of the *poePoweredDevice* parameters on the indicated port from its hardware. Call this command before calling *poePoweredDevice cget option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

poePoweredDevice set *chassisID cardID portID*

Sets the configuration of the *poePoweredDevice* parameters in IxHAL on port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration

option values set by the **poePoweredDevice config option value** command.
Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- The port is not a Packet over Sonet port.

poePoweredDevice **setDefault**

Sets to IxTclHal default values for all configuration options.

poePoweredDevice **setNominal *chasID cardID portID [controlType]***

Sets the option values for a particular control type to its nominal value.
controlType may be one of these:

Control Type	Usage
“class0”	Class 0 device emulation
“class1”	Class 1 device emulation
“class2”	Class 2 device emulation
“class3”	Class 3 device emulation
“class4”	Class 4 device emulation
“cpd”	AC MPS capacitance
“csig”	Detection signature capacitance
“idleCurrent”	Steady state current to idle current
“rpd”	AC MPS resistance
“rsig”	Detection signature resistance
“vClassify”	vClassify threshold voltage
“vDetect”	vDetect threshold voltage
“vNoop”	vNoop threshold voltage
“vOperate”	vOperate threshold voltage
“vOff”	vOff threshold voltage

EXAMPLES

```
package require IxTclHal

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```

# Get the chassis ID to use in port lists
set chId [ixGetChassisID $host]

set card 27
set portId 1
set waitForCalibration20

# Useful port lists
set portList [list [list $chassis $card $portId]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

set retCode "PASS"

poePoweredDevice setDefault

# Initiate an auto calibration
if {[poeAutoCalibration initiateCalibrate $chId $card $portId]} {
    errorMsg "Error initiating autoCalibrate on port $chId
$card $portId"
    set retCode "FAIL"
}
# Wait for the maximum possible time
after [expr $waitForCalibration*1000]

# Ask for the status of the calibration
if {[poeAutoCalibration requestStatus $chId $card $portId]} {
    errorMsg "Error requesting status on autoCalibrate for
port $chId $card $portId"
    set retCode "FAIL"
}
# Wait a bit for the answers to be read back
after 1000

if {[poeAutoCalibration get $chId $card $portId]} {
    errorMsg "Error getting status on autoCalibrate for port
$chId $card $portId"
    set retCode "FAIL"
    break
}

# Check to make sure that all calibrations succeeded.
foreach param {currentReadbackStatus iClassRangeStatus
iLoadRangeStatus \
iPulseRangeStatus voltageReadbackStatus} {
    if {[poeAutoCalibration cget -$param] !=
$poeAutoCalibrationPass} {
        errorMsg "poeAutoCalibration cget $param -
[poeAutoCalibration cget -$param] != poeAutoCalibrationPass"
        set retCode "FAIL"
    }
}

```

```

if {[poePoweredDevice get $chId $card $portId]} {
    errorMsg "Error getting poe config for $chId $card $portId"
    set retCode "FAIL"
    continue
}

if {[poePoweredDevice cget -enableClassSignature]} {
    switch [poePoweredDevice cget -classType] {
        $::poeClass0 {
            logMsg "PoE device config'd as class 0"
        }
        $::poeClass1 {
            logMsg "PoE device config'd as class 1"
        }
        $::poeClass2 {
            logMsg "PoE device config'd as class 2"
        }
        $::poeClass3 {
            logMsg "PoE device config'd as class 3"
        }
        $::poeClass4 {
            logMsg "PoE device config'd as class 4"
        }
        default {
            logMsg "PoE device in an in-between class state"
        }
    }
    # change the value if class enabled
    poePoweredDevice config -signatureValue 42.0
}

poePoweredDevice config -enableDetectionSignature true
poePoweredDevice config -rsig 25
poePoweredDevice config -csig 200.3
poePoweredDevice config -enableCsigAdd10uF false

poePoweredDevice config -enableAcMpsSignature true
poePoweredDevice config -rpdRangeControl poeRpdRangeZac1
poePoweredDevice config -rpd 33
poePoweredDevice config -cpd 42
poePoweredDevice config -enableCpdAdd10uF false

# config the steady state stuff
poePoweredDevice config -steadyStateLoadControl \
                      poeLoadControlControlledPower
poePoweredDevice config -controlledCurrent 482.2
poePoweredDevice config -controlledPower 13.8
poePoweredDevice config -idleCurrent 12.0

# config the transient load stuff
poePoweredDevice config -enableTransientLoadControl true
poePoweredDevice config -pulseWidth 10
poePoweredDevice config -enablePulseOnStart false
poePoweredDevice config -duty 33.3
poePoweredDevice config -pulsedCurrent 500
poePoweredDevice config -slewRate 20.0

# config the voltage threshold stuff
poePoweredDevice config -vOperate 27.5
poePoweredDevice config -vOff 32.8
poePoweredDevice config -vClassify 17.2
poePoweredDevice config -vDetect 7.9
poePoweredDevice config -vNoop 8.2

```

```

if { [poePoweredDevice set $chId $card $portId] } {
    errorMsg "Error setting poe config for \
              $chId $card $portId - $::ixErrorInfo"
}

# set nominal examples
if { [poePoweredDevice setNominal $chId $card $portId class0] } {
    errorMsg "Error setting nominal class0for \
              $chId $card $portId - $::ixErrorInfo"
}
if { [poePoweredDevice setNominal $chId $card $portId rsig] } {
    errorMsg "Error setting nominal rsig for \
              $chId $card $portId - $::ixErrorInfo"
}

ixWritePortsToHardware portList
ixLoadPoePulse portList

# signal acquisition
poeSignalAcquisition setDefault
poeSignalAcquisition config -enableTime      true
poeSignalAcquisition config -enableAmplitude true

poeSignalAcquisition config -startTriggerSource
poeTriggerSourceDCVolts
poeSignalAcquisition config -startTriggerSlope
poeTriggerSlopePositive
poeSignalAcquisition config -startTriggerValue      0.167

poeSignalAcquisition config -stopTriggerSource
poeTriggerSourceDCVolts
poeSignalAcquisition config -stopTriggerSlope
poeTriggerSlopePositive
poeSignalAcquisition config -stopTriggerValue      2.167

poeSignalAcquisition config -amplitudeMeasurementDelay 0.500

if [poeSignalAcquisition set $chId $card $portId] {
    errorMsg "Error setting poeSignalAcquisition for \
              $chId $card $portId $::ixErrorInfo"
}
if [ixArmPoeTrigger portList] {
    errorMsg "Error arming the PoE ports in the portList\
              $portList $::ixErrorInfo"
}

```

SEE ALSO*poeAutoCalibration, poeSignalAcquisition*

NAME - poeSignalAcquisition

poeSignalAcquisition - measure time period between PoE events

SYNOPSIS

poeSignalAcquisition *sub-command options*

DESCRIPTION

The **poeSignalAcquisition** command is used to set up and capture the time between two signal transition events. The amplitude of the a signal may also be measured a fixed time after the first signal transition.

The *startTriggerSource*, *startTriggerSlope* and *startTriggerValue* are used to indicate the signal to be used for the first event, the slope that it should transition (positive or negative) and the value that should be matched. Similarly, the *stopTriggerSource*, *stopTriggerSlope* and *stopTriggerValue* are used to indicate the signal to be used for the second event. The *enableTime*, *enableAmplitude* and *amplitudeMeasurementDelay* options are used to condition the measurements made.

Arming of the signal acquisition is accomplished through the use of the *portGroup* command with the *armPoeTrigger* value, or the *ixArmPoeTrigger* and *ixArmPortPoeTrigger* high-level commands. The arming may be aborted through the use of the *portGroup* command with the *abortPoeTrigger* value, or the *ixAbortPoeArm* and *ixAbortPortPoeArm* high-level commands.

A number of statistics available through the *stat*, *statGroup*, *statList*, and *statWatch* commands support operation of this command. The status of the arming may be read from the *statPoeTimeArmStatus* and *statPoeAmplitudeArmStatus* options. The status of the triggering may be read from the *statPoeTimeDoneStatus* and *statPoeAmplitudeDoneStatus* options. The time and amplitude values are visible in the *statPoeMonitorTime* and *statPoeMonitorAmplitudeDCVolts* and *statPoeMonitorAmplitudeDCAmps* options after a trigger has completed.

STANDARD OPTIONS

amplitudeMeasurementDelay

If *enableAmplitude* is *true*, then this value indicates the amount of time after the start trigger has been satisfied at which the amplitude measurement of the signal indicated in *startTriggerSource* is measured. Expressed in ms. (*default* =)

enableAmplitude *true* / *false*

If *true*, amplitude measurements is made. The amplitude measurement of the signal indicated in *startTriggerSource* is made *amplitudeMeasurementDelay* ms after the start trigger has been satisfied. (*default* =)

enableTime *true* / *false*

If *true*, then the time between the start trigger event and the stop trigger event is measured. (*default* =)

startTriggerSlope

Indicates which slope of the *startTriggerSignal* satisfies the start trigger event.

Option	Value	Usage
poeTriggerSlopePositive	0	(<i>default</i>) A positive slope.
poeTriggerSlopeNegative	1	A negative slope.

startTriggerSource

Indicates which signal characteristic is to be used to trigger the start event and to be measured if *enableAmplitude* is *true*.

Option	Value	Usage
poeTriggerSourceDCVolts	0	(<i>default</i>) DC voltage
poeTriggerSourceDCAmps	1	DC amperage

startTriggerValue

Indicates the value to be used as a threshold for the start trigger event. Expressed in volts or amps, depending on the setting of *startTriggerSource*. (*default* =)

stopTriggerSlope

Indicates which slope of the *stopTriggerSignal* satisfies the stop trigger event.

Option	Value	Usage
poeTriggerSlopePositive	0	(<i>default</i>) A positive slope.
poeTriggerSlopeNegative	1	A negative slope.

stopTriggerSource

Indicates which signal characteristic is to be used to trigger the stop event.

Option	Value	Usage
poeTriggerSourceDCVolts	0	(<i>default</i>) DC voltage
poeTriggerSourceDCAmps	1	DC amperage

stopTriggerValue

Indicates the value to be used as a threshold for the stop trigger event. Expressed in volts or amps, depending on the setting of *stopTriggerSource*. (*default* =)

COMMANDS

The *poeSignalAcquisition* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

poeSignalAcquisition cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *poeSignalAcquisition* command.

poeSignalAcquisition config option value

Modify the configuration options of the *poeSignalAcquisition*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *poeSignalAcquisition*.

poeSignalAcquisition get chasID cardID portID

Gets the options associated with a particular PoE port. Specific errors are:

- Port is not available

poeSignalAcquisition set chasID cardID portID

Sets the options associated with a particular PoE port. Specific errors are:

- No connection to the chassis
- Invalid port - not available or in use

poeSignalAcquisition setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *poePoweredDevice*

SEE ALSO

poePoweredDevice

NAME - port

port - configure a port of a card on a chassis.

SYNOPSIS

port *sub-command options*

DESCRIPTION

The **port** command is used to configure the properties of a port of a card on a chassis.

For Sonet ports which support DCC (Data Communications Channel) streams and flows, ports may be set transmit a combination of DCC packets as streams, advanced streams or flows and SPE (Synchronous Payload Envelope) packets as streams, advanced streams or flows. See the *transmitMode* option for further details.

Note: the *setDefault* sub-command sets all options at default values, as indicated here. These values are a consistent setting for 10/100 ethernet cards and may or may not be appropriate for other cards. In general, the sequence:

```
port setDefault
port set $chassis $card $port
fails.
```

The *setFactoryDefaults* sub-command, which relates to a particular port, sets all options at default values appropriate for the type of port. The sequence:

```
port setFactoryDefaults $chassis $card $port
port set $chassis $card $port
```

always succeeds. For multi-type boards, for example, OC192/10GE WAN, the board type is forced to one particular setting and may not be appropriate.

STANDARD OPTIONS

**advertise1000Full
Duplex *true/false***

If set to *true*, this port advertises itself at 1000 Mbps and Full duplex mode (*applicable to gigabit ports only*). (*default = false*)

**advertise100FullDuplex
*true/false***

If set to *true*, this port advertises itself at 100 Mbps and Full duplex mode (*applicable to 10/100 port only*). (*default = true*)

**advertise100Half
Duplex *true/false***

If set to *true*, this port advertises itself at 100 Mbps and Half duplex mode (*applicable to 10/100 port only*). (*default = true*)

**advertise10FullDuplex
*true/false***

If set to *true*, this port advertises itself at 10 Mbps and Full duplex mode (*applicable to 10/100 port only*). (*default = true*)

**advertise10HalfDuplex
*true/false***

If set to *true*, this port advertises itself at 10 Mbps and Half duplex mode (*applicable to 10/100 port only*). (*default = true*)

advertiseAbilities

Sets up the auto-negotiation parameters for gigabit (*applicable to Gigabit only*). The value of *flowControl* must be *true* for this field to have an effect.

Option	Value	Usage
portAdvertiseNone	0	(<i>default</i>) Do not advertise flow control abilities

Option	Value	Usage
portAdvertiseSend	1	Send only (asymmetric to link partner)
portAdvertiseSendAndReceive	2	Send and receive (symmetric to link partner)
portAdvertiseSendAndOrReceive	3	Send and/or receive (both symmetric or asymmetric to link partner)

**autoDetectInstrumentation
-tionMode**

For specified load modules, the timestamp can be inserted into the Auto Instrumentation header instead of the usual locations such as before CRC or at user-specified offset. This is called “Floating Timestamp.”
(default = portAutoInstrumentationModeEndOfFrame) Options include:

Option	Value	Usage
portAutoInstrumentationModeEndOfFrame	0	(default) End of Frame timestamp and data integrity
portAutoInstrumentationModeFloating	1	Floating timestamp and data integrity

**autonegotiate
*true/false***

Sets/unsets auto-negotiate mode on a 10/100 port. *(default = false)*
 Autonegotiate is always **true** for Gigabit ports.

dataCenterMode

Configure the type of priority traffic mapping on a port if *portFeatureDataCenterMode = true*. *(default = fourPriorityTrafficMapping)*

Option	Value	Usage
fourPriorityTrafficMapping	1	four priority
eightPriorityTrafficMapping	2	eight priority

DestMacAddress

The MAC address of the DUT port to which the Ixia source port is connected.
 Used for running IP tests. Entered in form {01 02 03 04 05 06}. *(default = {00 de bb 00 00 00})*

Note: This value is not written in HAL or hardware. It is merely stored in TclHal so that it can be accessed at any time. The MAC addresses should be set with the *stream* command.

directedAddress

This is the address that port listens on for a directed pause message. *(default = {01 80 C2 00 00 01})*

duplex half/full

Set the duplex mode to half duplex or full duplex on a 10/100 port. *(applicable to 10/100 only)* *(default = full)*

Duplex is always **full** for Gigabit ports.

**enableAutoDetect
Instrumentation *true/false***

If set to *true*, then auto detection of instrumentation is enabled. *(default = false)*

**enableDataCenterMode
*true/false***

Enable/disable the Data Center Mode. *(default = false)*

**enableManualAuto
Negotiate *true/false***

If set to *true*, then as the port configuration is written to hardware auto negotiation begins. *(applicable to MII only)* *(default = false)*

**enablePhyPolling
*true/false***

If set to *true*, the PHY is continuously polled during Mii setup operation. (*default = true*)

**enableRepeatableLastRandomPattern
*true/false***

This feature is only available for certain port types; this may be tested through the use of the *port is ValidFeature... portFeatureRepeatableRandomStreams* command. If *true*, the last random seed used to create random stream values is reused. That value is visible in the *lastRandomSeedValue* option. (*default = false*)

enableSimulateCableDisconnect *true / false*

If set to *true*, the port simulates a disconnected cable to the DUT. (*default = false*)

**enableTransparentDynamicRateChange
*true / false***

If set to *true*, the port allows dynamic rate change across counters. (*default = false*)

enableTxRxSyncStatsMode *true / false*

If set to *true*, the port collects Tx/Rx sync stats. (*default = false*)

flowControl *true/false*

Sets/unsets flow control on a port. (*default = false*)

flowControlType

Configure the type of flow control on a port if *portFeatureDataCenterMode = true*. (*default = ieee8023x*)

Option	Value	Usage
ieee8023x	0	(<i>default</i>) IEEE 802.3x values For details, see pauseControlType on page A-494.
ieee8021Qbb	1	IEEE 802.1Qbb values For details, see pauseControlType on page A-494.

gigVersion

Read-only. FPGA version of the gigabit port. (*applicable to Gigabit only*)

ignoreLink *true/false*

Transmit ignores the link status on Ethernet, POS or ATM port if set to *true*. (*default = false*)

lastRandomSeedValue

Read-only. The seed value that was last used when *enableRepeatableLastRandomPattern* was *false* and a start transmit operation was performed.

linkState

Read-only. The following states can be read from the port:

Option	Value	Usage
linkDown	0	The link on the port is down. This may be because there is no cable connected to the port or the link on the destination port may be down. The LED on the card is off when the link is down.
linkUp	1	the link is up indicated by green LED on the card.
linkLoopback	2	the port has been set to loopback mode. The LED on the card is off in this mode.
miiWrite	3	the link is in this state when the configuration of 10/100 port is being written to hardware (<i>applicable to 10/100 only</i>)
restartAuto	4	restarts the auto-negotiation process

Option	Value	Usage
autoNegotiating	5	the link is in currently executing the auto-negotiation process
miiFail	6	failed to write into memory for 10/100 ports (<i>applicable to 10/100 only</i>)
noTransceiver	7	No external transceiver or carrier detected.
invalidAddress	8	No PHY detected at the selected address.
readLinkPartner	9	Auto negotiation state in negotiation process. This is an intermediate state and should be used for informational purposes only.
noLinkPartner	10	Auto negotiation state in negotiation process. No link partner was found. This is an intermediate state and should be used for informational purposes only
restartAutoEnd	11	Auto negotiation state in negotiation process. This is an intermediate state and should be used for informational purposes only.
fpgaDownloadFail	12	Fpga download failure. Port is not usable.
noGbicModule	13	No GBIC module detected on Ixia Gbic port.
fifoReset	14	State in board initialization process. This is an intermediate state used for informational purposes only.
fifoResetComplete	15	State in board initialization process. This is an intermediate state and used for informational purposes only.
pppOff	16	PPP is disabled. PPP control packets is ignored; PPP link negotiation is not performed. Does not mean the link is unusable; it may, for instance, be configured for Cisco/HDLC and traffic (non-PPP) may still flow.
pppUp	17	The fully operational state when PPP is enabled. PPP link negotiation has successfully completed and the link is available for normal data traffic.
pppDown	18	The non-operational state when PPP is enabled. PPP link negotiation has failed or the link has been administratively disabled.
pppInit	19	PPP link negotiation state. <i>This is an intermediate state and should be used for informational purposes only</i> . Initialization state at the start of the negotiation process.
pppWaitForOpen	20	PPP link negotiation state: Waiting for indication from PPP controller that auto negotiation and related PPP control packet transfers can proceed. <i>This is an intermediate state and should be used for informational purposes only</i> .
pppAutoNegotiate	21	PPP link negotiation state: In process of exchanging PPP control packets (for example, LCP and IPCP) to negotiate link parameters. <i>This is an intermediate state and should be used for informational purposes only</i> .
pppClose	22	PPP link negotiation state: The PPP session has been terminated. All data traffic stops.

Option	Value	Usage
pppConnect	23	PPP link negotiation state: Negotiation has successfully completed; the peers are logically connected. Normal data traffic may flow once the pppUp state is reached. <i>This is an intermediate state and should be used for informational purposes only.</i>
lossOfFrame	24	Physical link is down. (for example, loss of signal, loss of frame)
lossOfSignal	25	Physical link is down. (for example, loss of signal, loss of frame)
lossOfFramePppDisabled	26	PPP link negotiation state: Physical link has gone down and PPP negotiation has been stopped.
stateMachineFailure	27	Communication with the local processor has failed. Check Server display and log for possible failure.
pppRestartNegotiation	28	PPP link negotiation state, following explicit request to restart negotiation process: this state indicates response to request. <i>This is an intermediate state and should be used for informational purposes only.</i>
pppRestartInit	29	PPP link negotiation state, following explicit request to restart negotiation process: the link has or is brought down to begin a new negotiation cycle. <i>This is an intermediate state and should be used for informational purposes only.</i>
pppRestartWaitForOpen	30	PPP link negotiation state, following explicit request to restart negotiation process: Waiting for indication from PPP controller that current connection is already down or is in process of being shut down. <i>This is an intermediate state and should be used for informational purposes only.</i>
pppRestartWaitForClose	31	PPP link negotiation state, following explicit request to restart negotiation process: Waiting for indication from PPP controller that shut down of current connection has completed. <i>This is an intermediate state and should be used for informational purposes only.</i>
pppRestartFinish	32	PPP link negotiation state, following explicit request to restart negotiation process: Preparation for restart completed; ready to begin normal cycle again. <i>This is an intermediate state and should be used for informational purposes only.</i>
localProcessorDown	33	local processor boot failure
forcedLinkUp	34	Link has been forced up.
temperatureAlarm	35	An over-temperature condition has occurred.
pppClosing	36	PPP negotiation is closing.
pppLcpNegotiate	37	PPP LCP negotiation in process.
pppAuthenticate	38	PPP authentication in process.
pppNcpNegotiate	39	PPP NCP negotiation in process.
noXenpakModule	40	No Xenpak module is installed.
sublayerUnlock	41	Sublayer unlock.
demoMode	42	Server is in demo mode.
waitingForFpgaDownload	43	Port is waiting for FPGA (Field Programmable Gate Array) programming to be downloaded to port.
lossOfCell	44	ATM cell loss.

Option	Value	Usage
noXFPModule	45	No XFP module is installed.
moduleNotReady	46	The XFP interface has reported not ready.
noX2Module	48	No X2 module is installed.
lossOfPointer	49	Loss of pointer.
lossOfAlignment	50	Loss of alignment.
lossOfMultiframe	51	Loss of multiframe.
gfpOutOfSync	52	GFP out of sync.
lcasSequenceMismatch	53	Lcas sequence mismatch.
ethernetOamLoopback	54	Ethernet OAM loopback

loopback

Sets/unsets loopback mode on a port. (*default = portNormal*) Valid choices are:

Option	Value	Usage
portNormal	0	
portLoopback	1	
portLineLoopback	2	

MacAddress

Assigns a Source MAC address to the port. MAC address is entered in form {01 02 03 04 05 06}. (*default = '00 de bb 00 01 01'*)

Note: This value is not written in HAL or hardware. It is merely stored in TclHal so that it can be accessed at any time. The MAC addresses should be set with the *stream* command.

managerIp

Read-only. For ports with local CPUs, this is the management IP address associated with the port. For example, the default *managerIp* for port 1 on card 2 would be 10.0.2.1.

masterSlave

Only apply to GIG MII. If *negotiateMasterSlave* is ‘false’, then the **masterSlave** is essentially read-only. Options include:

Option	Value	Usage
portMaster	0	
portSlave	1	(<i>default</i>)

multicastPauseAddress

This is the address that the port listens on for a multicast pause message. (*default = {01 80 C2 00 00 01}*)

name

The given name of the port. (*default = “”*)

negotiateMasterSlave *true/false*

Only apply to Gigabit MII. Enable *negotiateMasterSlave*. (*default = false*)

numAddresses

Number of source MAC addresses assigned to this port. (*default = 1*)

Note: This value is not written in HAL or hardware. It is merely stored in TclHal so that it can be accessed at any time.

operationModeList

Use to configure port operation mode, for load modules with this option. Options include:

Option	Value	Usage
portOperationModeStream	0	(default) Sets port operation mode to Stream/Capture/Latency mode.
portOperationModeRtp	1	Sets port operation mode to RTP.
portOperationModeTsoLro	2	Sets port operation mode to TSO/LRO.
portOperationModeL7	3	Sets port operation mode to L7.
portOperationModeHWIPsec	4	Sets port operation mode to IPsec of hardware.

owner

Read-only. Name of the owner of this port, if any. (*default* = "")

packetFlowFileName

Sets the packet flow file name. To set the packet flow file name, need to enable *usePacketFlowImage File* first. (*default* = "")

pfcEnableValueList

Valid when *flowControlType* is set to *ieee8021Qbb*.

Use to configure priority-based flow control (PFC) with pair list of enable and channel mask value. (*default* = '{0 0} {0 0} {0 0} {0 0} {0 0} {0 0} {0 0} {0 0}') The first item in each pair is 'enable' and the second item is 'channel mask value'.

pfcResponseDelay Enabled

If true, sets the delay time, in nanoseconds, of frames.

pfcResponseDelay Quanta

Allows to set the delay quanta of flow control.

pfcEnableValueListBit Matrix

Valid when *flowControlType* is set to *ieee8021Qbb*.

Use to configure priority-based flow control (PFC) with pair list of enable and channel mask value. (*default* = '{0 0} {0 0} {0 0} {0 0} {0 0} {0 0} {0 0} {0 0}') The first item in each pair is 'enable' and the second item is 'channel mask value'.

pmaClock

(*default* = *pmaClockAutoNegotiate*) Options include:

Option	Value	Usage
pmaClockAutoNegotiate	0	Auto Negotiate
pmaClockMaster	1	(default) Transceiver 10G Base-T Phy Master
pmaClockSlave	2	Transceiver 10G Base-T Phy Slave

preEmphasis

For ports that support the *portFeaturePreEmphasis*, the percentage signal pre-emphasis to be applied. If a port does not support the exact percentage set in this option, the nearest value is used. Refer to the *Ixia Hardware Guide* for the exact pre-emphasis percentages supported. (*default* = 0)

phyMode

Read-only. The current PHY mode for cards which support both Copper and Fiber PHY modes. The current mode may be set with the *setPhyMode* sub-command.

Option	Value	Usage
portPhyModeCopper	0	(default) Copper
portPhyModeFiber	1	Fiber

portMode

Multimode ports may be set into one of their possible modes by setting this option. The setting of this option has no meaning for ports that only operate in a single mode. The speed of ports which operate at multiple speed is controlled by the *autonegotiate*, *speed*, *advertisexxx* and *duplex* options.

The choices for this option are:

Option	Value	Usage
portPosMode	0	(default) Packet over sonet mode.
portEthernetMode	1	Indicates Ethernet mode or 10Gig WAN mode.
port10GigWanMode		
port10GigLanMode	4	Indicates 10Gig LAN mode.
portBertMode	5	Indicates BERT mode.
portAtmMode	7	Indicates ATM mode.
portPosChannelizedMode	8	Indicates Channelized POS mode

The valid choices for OC48c POS/BERT and OC48cTXS POS/BERT combinations are:

Option	Value	Usage
portPosMode	0	POS mode.
portBertMode	5	BERT mode.

The valid choices for OC192c POS/WAN/BERT are:

Option	Value	Usage
portPosMode	0	POS mode.
port10GigWanMode	1	10Gig WAN mode.
portBertMode	5	BERT mode.

Earlier values of *portPosFraming* and *posEthernetFraming* are still valid and produces the same results as the use of *portPosMode* and *portEthernetMode*, but are deprecated for future use.

The valid choices for 10GE POS/WAN/LAN/BERT are:

Option	Value	Usage
portPosMode	0	POS mode.
port10GigWanMode	1	10Gig WAN mode.
port10GigLanMode	4	10Gig LAN mode.
portBertMode	5	BERT mode.

Note that *port setFactoryDefault* will not reset the port mode associated with OC192/10Gig type cards.

The valid choices for ATM/POS are:

Option	Value	Usage
portPosMode	0	POS mode.
portAtmMode	7	ATM mode.
portPosChannelizedMode	8	POS channelized mode

receiveMode

Sets up the type of capture/receive mode for this port.

Note: The receive modes are and'd and or'd to determine which fpga is required for what interface. If a port does not support receiveMode, then any of these options that are configured has no effect. Options include:

Option	Value	Usage
portRxModeNone	0	The displayed value for ports that do not support receive mode. Using this option for ports that DO support receiveMode has no effect.
portCapture	0x0001	(default) use normal capture buffer
portPacketGroup	0x0002	get real time latency on received packets
portRxTcpSessions	0x0004	use TCP session
portRxTcpRoundTrip	0x0008	do TCP Round trip
portRxDataIntegrity	0x0010	do data integrity
portRxFirstTimeStamp	0x0020	get the first receive time
portRxSequenceChecking	0x0040	do sequence checking
portRxModeBert	0x0080	Bit Error Rate testing mode
portRxModeIsl	0x0100	Expect ISL encapsulation
portRxModeBertChannelized	0x0200	Channelized BIT Error rate testing mode
portRxModeEcho	0x0400	Gigabit echo mode
portRxModeDcc	0x0800	DCC packets are received from the SONET overhead.
portRxModeWidePacketGroup	0x1000	Latency mode using wide packet groups
portRxModePrbs	0x2000	Enable capture of PRBS packets Note: Wide packet group must be enabled when using PRBS. Note: When selected, if Data Integrity was previously selected, it is disabled and a message logs to the Tcl event log to note the change in the receive mode.
portRxModeRateMonitoring	0x4000	Enable capture of Rate Monitoring packets Note: Wide packet group must be enabled when using Rate Monitoring. Note: When selected, if Sequence Checking was previously selected, it is disabled and a message logs to the Tcl event log to note the change in the receive mode.
portRxModePerFlowErrorStats	0x8000	Enables capture of per-PGID checksum error stats. Note: When selected, Wide Packet Groups is automatically enabled.

rxFpgaVersion *Read-only*. FPGA version of the receive engine of the 10/100 port. (*applicable to 10/100 only*)

rxTxMode Sets one of following modes on a Gigabit port

Option	Value	Usage
gigNormal	0	(<i>default</i>) The Gigabit port runs as full duplex.
gigLoopback	1	The Gigabit port transmits and receives frames in internal loopback.
gigCableDisconnect	2	simulate cable disconnect on the port

speed 10/100/1000

Set the line speed in MBps. Note that this value does not represent an actual line rate. Some deprecated older commands needed this value to perform various operations. New commands no longer need the value. When using the *cget* command to return the value, the value return will not reflect the actual line rate. (*default = 100*)

timeoutEnable
true / false

Enables the gigabit auto-negotiation timeout. (*applicable to Gigabit only*)
(*default = true*)

transmitClockDeviation

For ports that support the *portFeatureFrequencyOffset* feature, this is the transmit clock deviation expressed in parts per million (ppm). (*default = 0*)

transmitClockMode

Configure the type of clock mode on a port. (*default = portClockInternal*)

Option	Value	Usage
portClockInternal	0	internal clock
portClockExternal	1	external clock

transmitMode

Sets the type of stream/transmit mode for this port. Options include:

Option	Value	Usage
portTxPacketStreams	0	(<i>default</i>) set up hardware to use normal streams
portTxPacketFlows	1	set up hardware to use packet flows
portTxModeAdvancedScheduler	4	set up hardware to use the advanced scheduler
portTxModeBert	5	set up the hardware to use Bit Error Rate patterns
portTxModeBertChannelized	6	set up the hardware to use channelized BERT
portTxModeEcho	7	sets up port to echo received packets
portTxModeDccStreams	8	sets up the port to only transmit DCC packets as a stream
portTxModeDccAdvancedScheduler	9	sets up the port to only transmit DCC packets as advanced streams
portTxModeDccFlowsSpeStreams	10	sets up the port to transmit DCC packets as flows and SPE packets as streams
portTxModeDccFlowsSpeAdvancedScheduler	11	sets up the port to transmit DCC packets as flows and SPE packets as advanced streams

A**port**

Option	Value	Usage
portTxModeAdvancedScheduler Coarse (VM only)	12	set up VM to use the advanced scheduler with less precision and cpu utilization
portTxModePacketStreamsCoarse(VM only)	13	set up VM to use the streams with less precision and cpu utilization

txFpgaVersion

Read-only. FPGA version of the transmit engine of the 10/100 port. (*applicable to 10/100 only*)

txRxSyncInterval

The interval (ms) at which to synchronously freeze TX and RX PGID stats.

type

Read-only. Specifies the type of the Ixia port. The following options are used, along with the name of the port found when using IxExplorer. The Ixia part number associated with each port type can be found in the *Ixia Hardware Guide*.

Option	Value	IxExplorer Port Name
port10100BaseTX	1	10/100 Base TX
port10100BaseMII	2	10/100 MII
port100BaseFXMultiMode	3	100 Base FX MultiMode
port100BaseFXSingleMode	4	100 Base FX SingleMode
portGigabitSXMultimode	5	1000 Base SX MultiMode
portReducedMII	7	10/100 Reduced MII
portGbic	8	GBIC
portPacketOverSonet	9	OC12c/OC3c POS
port10100Level3	10	10/100 Base TX - 3
portGigabitLevel3	11	1000 Base SX MultiMode - 3
portGbicLevel3	12	GBIC-3
portGigCopper	13	GBIC
portPosOc48	14	OC48c POS
portPosOc48Level3	15	OC48c POS-M
portPosOc192	16	OC192c POS
portPosOc192Level3	17	OC192c POS-3
portPosOc48VariableClocking	27	OC48c POS VAR
portGigCopperTripleSpeed	28	Copper 10/100/1000
portGigSingleMode	29	1000 Base LX SingleMode
portOc48Bert	32	OC48c POS BERT
portOc48PosAndBert	33	OC48c POS/BERT
port10GEWAN2	36	OC192c POS
port10GEWAN1	37	OC192c POS OC192c VSR OC192c POS/BERT/10GE WAN 10GE BERT/WAN
port10GEXAUI1	45	10GE XAUI 10GE XAUI/BERT 10GE XAUI BERT

Option	Value	IxExplorer Port Name
port10GigLanXenpak1	49	10GE XENPAK 10GE XENPAK-M 10GE XENPAK/BERT 10GE XENPAK BERT 10GE XENPAK-MA/BERT
port10GELAN_M	51	
port10GELAN1	53	10GE LAN 10GE LAN-M
port10100Txs	63	10/100 Base TX
port1000Sfps4	67	1000 Base X 1000 Base X L7
port1000Txs4	68	10/100/1000 Base T 10/100/1000 Base T (L7)
portSingleRateBertUnframed	69	Unframed BERT Single-Rate
portMultiRateBertUnframed	70	Unframed BERT Multi-Rate
port10GEUniphy_MA	71	
port10GEUniphy	72	10GE LAN/WAN / OC192c POS/BERT
port40GigBertUnframed	73	Unframed Bert 40Gig Port
portOc12Atm	74	ATM 622 Mutli-Rate
portOc12Pos32Mb	75	OC12 POS 32MB
port1000Txs24	77	10/100/1000 Base T
portElm	78	
port101001000Layer7	80	
port10GEXenpakP	81	
port1000Stxs4	82	
port10GUuniphyP	83	
port10GELSM	84	
port10GEMultiMSA	85	
port10GUuniphyXFP	86	
portPowerOverEthernet	87	Power over Ethernet
port2Dot5GMSM	88	POS
port10GMSM	89	POS LAN/WAN
port101001000Inline	90	10/100/1000 Base T - Inline
port101001000Monitor	91	10/100/1000 Base T - Monitor
portASM101001000XMV12X	94	10/100/1000 ASM XMV12X
portASMXMV10GigAggregated	95	10G LAN XFP Aggregate
portLANXFP	97	10G LAN/WAN XFP (MACSec)
port10GLANWANXFP	98	10GE LSM XM8
portVoiceQualityResourceModule	99	Voice quality resource module
port40GE100GELSM	100	40GE LSM XMV and 100GE LSM XMV modules
portFlexAP10G16S	102	10G, 16-port Excellon-Flex port
port40GELSMQSFP	104	40 GE LSM QSFP port
portFCMSFP	105	4 and 8 port Fibre Channel with SPF+ interface

A**port**

Option	Value	IxExplorer Port Name
portEthernetVM	107	Ethernet VM port

typeName

Read only. The name equivalent of the *type* field.

**usePacketFlowImage
File *true/false***

Enable the Packet Flow Image File. Controls whether the port is used in stream mode or flow mode. If set to flow mode, then the *packetFlowFileName* option should be set. (*default =false*)

**DEPRECATED
STANDARD OPTIONS****dataScrambling**

Enables port data scrambling.

lineScrambling

Enables line scrambling.

rateMode

The rate may be entered in one of the following modes. Note: This value is not written in HAL or hardware. It is merely stored in TclHal so that it can be accessed at any time.

Option	Value	Usage
useGap	0	The rate is entered in clock ticks used to calculate the inter-frame gap
usePercentRate	1	the rate is entered as a percentage of maximum rate

sonetInterface**sonetOperation****useRecoveredClock
*true/false***

Set the sonet framer to use the recovered clock. (*applicable to POS/sonet only, non-LSM modules. LSM modules configure their recovered clock in the XAUI object.*) (*default =false*)

portMode

The following *portMode* options have been deprecated::

Option	Value	Usage
portUsbMode	2	Indicates USB mode for USB/Ethernet ports.

type

The following *type* options have been deprecated::

Option	Value	Usage
portUsbUsb	18	USB
portUsbEthernet	20	Ethernet
port10100UsbSh4	55	

COMMANDS

The **port** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

port `canUse` *chasID cardID portID*

If the port is owned by the current logged in user, `canUse` returns *true*, otherwise it returns *false*. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

port `cget` *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **port** command.

port `config` *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

CAUTION: 'Port get' should be called before 'port config' in order to maintain consistency between Tcl Client port object and Server port object.

port `export` *fileName chasID cardID portID*

Exports the current configuration of the port at *portID*, *cardID*, *chasID* into the file named *fileName*; *fileName* may include a full or relative path. The file produced by this command may be used by the *import* sub-command. Specific errors are:

- No connection to a chassis
- Invalid port

port `get` *chasID cardID portID*

Gets the current configuration of the port with id *portID* on card *cardID*, chassis *chasID* from its hardware. Call this command before calling **port cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

port `getFeature` *chasID cardID portID featureList*

Determines whether a specific feature is present in the *featureList* for the port at *portID*, *cardID*, *chasID*. A value list with unit is returned if the feature(s) are present; otherwise, an empty string is returned.

Feature	Request String	Description
Ethernet Line Rate	ethernetLineRate	Available ethernet line rates are returned as a list of numbers. Unit of measurement is mbit. An empty list is returned for single rate and Sonet boards. Example: { {10 100 1000} mbit}
Sonet Interface Type	sonetInterfaceType	Available Sonet interfaces are returned as a list of symbols. An empty list is returned for non-Sonet boards. Available symbols: oc3, oc12, oc48, stm1c, stm4c, stm16c, oc192, oc64c, ethOverSonet, ethOverSdh. Example: { oc3 oc12 }
Capture Buffer Size	captureBufferSize	The size of the capture buffer is given in MB as a float. Returns empty list for boards that do not support capture. Example: { 256.0 MB }

Feature	Request String	Description
Minimum Captured Packet Size	minimumCapturedPacketSize	The minimum number of bytes a packet must have to be captured. Returns empty list for boards that do not support capture. Example: { 12 bytes }
Maximum Captured Packet Size	maximumCapturedPacketSize	The maximum number of bytes a packet can have to be captured. Returns empty list for boards that do not support capture. Example: { 1518 bytes }
Number of Streams	basicStreamCount	The number of non-advanced streams the board supports is returned as an integer. Returns empty list for boards that do not support streams. Example: { 128 }
Number of Advanced Streams	advancedStreamCount	The number of streams the board supports is returned as an integer. Returns empty list for boards that do not support streams. Example: { 16 }
Minimum Preamble Size	minimumPreambleSize	The minimum number of preamble bytes is returned. Boards or modes that do not support this concept return an empty list. Unit of measurement is bytes. Example: { 2 bytes } { }
Maximum Preamble Size	maximumPreambleSize	The maximum number of preamble bytes is returned. Boards or modes that do not support this concept return an empty list. Unit of measurement is bytes. Example: { 12 bytes }
Minimum Frame Size	minimumFrameSize	The minimum number of bytes in a frame is returned. Unit of measurement is bytes. This includes CRC but not preamble bytes. Example: { 64 bytes }
Maximum Frame Size	maximumFrameSize	The maximum number of bytes in a frame is returned. Unit of measurement is bytes. This includes CRC but not preamble bytes. Examples: { 1516 bytes } { 65520 }
Minimum Inter-Frame Gap	minimumInterFrameGap	The minimum inter-frame gap is returned as a float. Unit of measurement is ns. Example: { 36.3 ns }
Maximum Inter-Frame Gap	maximumInterFrameGap	The maximum inter-frame gap is returned as a float. Unit of measurement is ns. Example: { 893621.3 ns }
Minimum Inter-Burst Gap	minimumInterBurstGap	The minimum inter-burst gap is returned as a float. Unit of measurement is ns. Example: { 36.3 ns }
Maximum Inter-Burst Gap	maximumInterBurstGap	The maximum inter-burst gap is returned as a float. Unit of measurement is ns. Example: { 893621.3 ns }
Minimum Inter-Stream Gap	minimumInterStreamGap	The minimum inter-stream gap is returned as a float. Unit of measurement is ns. Example: { 36.3 ns }
Maximum Inter-Stream Gap	maximumInterStreamGap	The maximum inter-stream gap is returned as a float. Unit of measurement is ns. Example: { 893621.3 ns }

Feature	Request String	Description
Minimum Frame Rate	minimumFrameRate	The minimum frame rate is returned as a float. Unit of measurement is fps. Example: { 0.321 fps }
Latency Resolution	latencyResolution	The resolution of port to port latency measurements using FPGA timestamps is returned as a float. The value is given for directly connected boards of the same type in the same chassis. Unit of measurement is ns. Example: { 40.0 ns }
Number of Virtual Circuits	virtualCircuitCount	The number of virtual circuits is returned. Boards that do not support this return an empty list. Examples: { 64 }
Phy Modes	phyModes	The list of available phy modes is returned as a list of symbols. Available symbols are portPhyModeCopper and portPhyModeFiber. Boards that do not have phy modes return an empty list. Example: { portPhyModeCopper portPhyModeFiber }
Total Port CPU Memory	totalPcpuMemory	The total Port CPU memory is returned. Example: { 256 }
Number of Table UDF Entries	tableUdfEntryCount	The maximum number of table UDF entries is returned. Boards that do not support this feature return an empty list. Example: { 10000 }
Number of Value List UDF Entries	valueListUdfEntryCount	The maximum number of value list UDF entries is returned. Boards that do not support this feature return an empty list. Examples: { 10000 }
Number of Range List UDF Entries	rangeListUdfEntryCount	The maximum number of range list UDF entries is returned. Boards that do not support this feature return an empty list. Examples: { 10000 }
Number of PGIDs	pgidCount	The maximum number of PGIDs returned. Boards that do not support this feature return an empty list. Example: { 10000 }
Number of Random Table Counts	randomTableCount	Returns the result of GetRandomTableCount(). Boards that do not support this feature return an empty list. Examples: { 4 }
Maximum UDF Count	maximumUdfCount	The maximum number of UDFs configured per port.

EXAMPLE

```

package req IxTclHal
ixConnectToChassis loopback
(TclScripts) 4 % port getFeature 1 177 1 {maximumUdfCount }
{maximumUdfCount {{ 8 }}}
(TclScripts) 5 % port getFeature 1 1 1 {maximumUdfCount }
{maximumUdfCount {{ 5 }}}
(TclScripts) 6 % port get 1 177 1
0
(TclScripts) 7 % port cget -type
106
(TclScripts) 8 % port cget -typeName
10GE LAN SFP+
(TclScripts) 9 % card get 1 177
0
(TclScripts) 10 % card cget -typeName
XDM10G32S

```

```
(TclScripts) 11 % card cget -type
170
port getFeature 1 1 1 {maximumUdfCount tableUdfEntryCount}
Returns {maximumUdfCount {{ 5 }} } {tableUdfEntryCount {98048} }
```

port getId *chasID cardID portID*

Gets the name of the port as a string of format <chassis>.<card>.<port> <portname> or <chassis>.<card>.<port> if the port has no name. For example, 1.2.3 router1.

port getPortState *chasID cardID portID*

Gets the ownership state of the port as a string of format <chassis>.<card>.<port> <portname> or <chassis>.<card>.<port> if the port has no name. For example, 1.2.3 router1.

port getStreamCount *chasID cardID portID*

Gets the number of streams configure on the with id <chassis>.<card>.<port> <portname> or <chassis>.<card>.<port> if the port has no name. For example, 1.2.3 router1.

port import *fileName chasID cardID portID*

Imports a saved port configuration found in the file *fileName* into the current configuration of the port at *portID*, *cardID*, chassis *chasID*. *fileName* may include a full or relative path. The file used by this command must have been produced by the *export* sub-command. Do not call a *port set* command after calling *port import* until a *port write* command is called. A *port write* is necessary to commit these items to the hardware. Specific errors are:

- No connection to a chassis
- Invalid port
- The card is owned by another user
- *fileName* does not exist

port isActiveFeature *chasID cardID portID feature*

Determines whether a specific feature is active for the port at *portID*, *cardID*, chassis *chasID* and that the port is properly configured/enabled to use that feature. A value of true (1) is returned if the feature is enabled and false (0) if the feature is not enabled. *Feature* may be one of the values from the *isValidFeature* list.

port isCapableFeature *chasID cardID portID feature [param]*

Determines whether a specific feature is capable for the port at *portID*, *cardID*, chassis *chasID*. A value of true (1) is returned if the port is capable of the feature and false (0) if not. *Feature* may be one of the values from the *isValidFeature* list.

port isValidFeature *chasID cardID portID feature [param]*

Determines whether a specific feature is valid for the port at *portID*, *cardID*, chassis *chasID* with the port in its current mode (for example, BERT versus LAN mode). A value of true (1) is returned if the feature is valid and false (0) if the feature is invalid or the port is invalid. The *param* option allows further

clarification on the *feature*; see the table below to determine the use of *param* for a particular feature. *Feature* may be one of the following values:

Feature	Value	Description
	0	Invalid feature
portFeatureQos	1	QoS statistics available.
portFeaturePacketFlows	2	supports packet flow mode.
portFeatureUdfOddOffset	3	UDFs can occur at odd byte offsets. <i>param</i> can be used as a particular UDF number to determine if the feature is supported on a particular UDF.
portFeatureRxPacketGroups	4	supports packet group mode.
portFeatureRxSequence Checking	5	supports receive sequence checking.
portFeatureRxDataIntegrity	6	supports received data integrity checking.
portFeatureRxRoundTripFlows	7	supports receive round trip flow mode.
portFeatureGigGMiiAutoDisable	8	Reserved for future usage.
portFeatureMultipleDLCIs	9	supports ability to generate multiple DLCIs per port on frame relay connections
portFeatureForcedCollisions	10	supports forced collisions.
portFeatureTxDataIntegrity	11	supports transmitted data integrity.
portFeaturePacketFlowImage File	12	supports packet flow mode with image files as described in the port command (this one).
portFeatureSrp	13	supports the spatial reuse protocol.
portFeaturePos	14	supports packet over sonet operation
portFeatureBert	15	supports bit error rate testing.
portFeature10GigWan	16	supports 10 Gbps Wide Area Network operation.
portFeature10GigWanAndOc192	17	supports 10 Gbps Wide Area Network, OC192 and BERT
portFeature10GigWanAndOc192	18	supports 10 Gbps Wide Area Network and OC192
portFeature10GigWanAndBert	19	supports 10 Gbps Wide Area Network and Bert
portFeatureOc192AndBert	20	supports OC 192 and Bert
portFeatureOC192Bert	21	supports OC192 Bert only
portFeatureUdfOverlap	22	multiple UDFs may start adjacent to each other, regardless of word boundaries. <i>param</i> can be used as a particular UDF number to determine if the feature is supported on a particular UDF.
portFeatureUdfCascade	23	supports cascading of user defined fields. <i>param</i> can be used as a particular UDF number to determine if the feature is supported on a particular UDF.
portFeatureRxSequence CheckingPerPGID	24	Sequence checking of packets with identical PGIDs is allowed
portFeaturePacketStreams	25	supports packet streams.
portFeatureAdvancedScheduler	26	supports advanced stream schedule operation.

Feature	Value	Description
portFeatureProtocols	27	supports operation of the protocol server.
portFeatureProtocolARP	28	supports ARP operation.
portFeatureProtocolPING	29	supports ping operation.
portFeatureBitMask	30	a bit mask may be used on UDF values without restriction; some boards require that bits in the mask be contiguous. <i>param</i> can be used as a particular UDF number to determine if the feature is supported on a particular UDF.
portFeatureSonetErrorInsertionList	31	supports insertion of sonet errors
portFeatureBertErrorGeneration	32	support Bert error generation
portFeatureLocalCPU	35	supports a local CPU
portFeatureIxRouter	36	can run a copy of IxRouter - used in IxLoad and other TCP level testing
portFeatureIxWeb	37	can be used in IxWeb
portFeature10GigLan	38	supports 10 gigabit ethernet LAN operation
portFeatureVsr	39	supports OC192 VSR operation
portFeatureSplitUdfs	40	UDFs may be split in combinations of 8, 16 and 32 bit counters. <i>param</i> can be used as a particular UDF number to determine if the feature is supported on a particular UDF.
portFeatureTxDuration	41	supports the Transmit duration statistic
portFeatureRxFirstTimeStamp	43	supports First Time Stamp operation
portFeatureRxStreamTrigger	44	supports stream trigger operation
portFeatureRxChecksumErrors	45	supports receive checksum operation
portFeatureOddPreamble	46	supports an odd number of bytes in the preamble
portFeaturePacketGapTimeUnits	47	supports different units of time in the packet gap specification (<i>gapUnit</i> in <i>stream</i>).
portFeatureRoutingProtocols	48	supports the advanced routing protocols
portFeatureModifiablePreamble	52	allows packet preamble contents to be modified
portFeatureIgnorePGIDSignature	77	allows the PGID signature to be ignored in latency measurements
portFeatureBertUnframed	82	supports unframed BERT
portFeatureXauI	84	10GigE XAUIC interface
portFeatureBertChannelized	92	Channelized BERT
portFeatureLdp	96	supports LDP operation
portFeatureUdf5	104	supports 5 UDFs.
portFeatureTxDccStreams	110	supports transmission of DCC packets as streams
portFeatureTxDccAdvancedScheduler	111	supports transmission of DCC packets as advanced scheduler streams.
portFeatureTxDccFlowsSpeStreams	112	supports transmission of a combination of DCC packets as flows and SPE packets as streams.

Feature	Value	Description
portFeatureTxDccFlowsSpeAdvancedScheduler	113	supports transmission of a combination of DCC packets as flows and SPE packets as advanced scheduler streams.
portFeatureRxDcc	114	supports reception of DCC packets.
portFeatureDccProperties	115	supports DCC features
portFeatureProtocolL2VPN	119	supports Layer 2 VPN
portFeatureProtocolL3VPN	120	supports Layer 3 VPN
portFeatureProtocolRIPng	121	supports RIPng protocol
portFeatureSrpFullFeatured	122	supports all SRP features
portFeatureUdfExtension1	123	supports advanced UDF extensions, including nested UDFs. <i>param</i> can be used as a particular UDF number to determine if the feature is supported on a particular UDF.
portFeatureTxFrequencyDeviation	131	supports the ability to vary the transmit frequency
portFeatureDaCascadeFromSelf	133	supports the ability to cascade a stream Destination MAC address from itself
portFeatureUdfTableMode	136	supports value list mode UDFs. <i>param</i> can be used as a particular UDF number to determine if the feature is supported on a particular UDF.
portFeatureCapture	143	supports received data capture.
portFeaturePauseControl	147	supports automatic pause control
portFeatureCJPAT	149	support for CJPAT jitter test pattern
portFeatureCRPAT	150	support for CRPAT jitter test pattern
portFeatureProtocolIGMP	151	supports the newer IGMP protocol implementation, which includes IGMPv3 and MLD
portFeatureAtm	152	the port is an ATM type
portFeatureRpr	153	support for RPR ring control signalling
portFeatureLinkFault	159	support for link fault signalling
portFeatureProtocolMLD	160	supports the MLD protocol
portFeatureProtocolPIMSM	163	support for the PIM/SM protocol
portFeatureProtocolOSPFv3	164	support for the OSPFv3 protocol
portFeatureProtocolIPv6NeighborDiscovery	165	supports the IPv6 neighbor discovery protocol
portFeatureProtocolBGPv6	166	supports BGP for IPv6
portFeatureProtocolISISv6	167	supports ISIS for IPv6
portFeatureFlexibleTimestamp	168	support flexible time stamp placement
portFeatureProtocolOffset	169	supports flexible placement of start of protocol in a frame
portFeatureRandomGap	171	support random gap values
portFeatureScheduledTxDuration	173	support setting of the maximum transmission time. See <i>portGroup setScheduledTxDuration</i> .
portFeatureLayer7Only	174	only supports Layer 7 operations. Such ports have no capabilities that can be used by the TCL API.

Feature	Value	Description
portFeatureUniphy	175	card is an OC192 type which supports WAN/LAN features simultaneously
portFeatureUdfIPv4Mode	176	support UDF in IPv4 mode. <i>param</i> can be used as a particular UDF number to determine if the feature is supported on a particular UDF.
portFeatureRandomFrameSizeWeightedPair	180	supports random weighted frame sizes
portFeatureRxWidePacketGroups	181	supports wide packet groups
portFeatureDualPhyMode	182	the ports on the card can operate in copper or fiber mode
portFeatureAtmPos	184	supports POS over ATM.
portFeatureFec	187	supports FEC (Forward Error Correction) operation in the optical digital wrapper
portFeatureAtmPatternMatcher	190	supports filter pattern matching for ATM patterns
portFeatureGfp	192	support GFP (Generic Framing Protocol) operation.
portFeatureCiscoCDL	198	supports Cisco CDL (Converged Data Layer) operation.
portFeatureRxLatencyBin	200	supports latency bins in packetGroups.
portFeatureRxTimeBin	201	supports time bins in packetGroups.
portFeaturePreambleView	204	supports the ability to view a preamble in <i>stream packetView</i> .
portFeaturePreambleCapture	205	supports the ability to capture a received preamble.
portFeatureCDLErrorTrigger	207	supports the ability to trigger capture from the presence of a CDL error.
portFeatureSimulateCableDisconnect	209	supports the ability to simulate a cable disconnect on the interface.
portFeatureTableUdf	211	supports table mode UDFs.
portFeatureOc192	212	supports OC192 operation.
portFeaturePerStreamTxStats	215	support per stream transmit statistics
portFeatureLasi	216	a XENPAK port that supports LASI operation
portFeatureIPsecAcceleration	218	a port that supports IPsec operation.
portFeaturePowerOverEthernet	219	supports PoE power consumption
portFeatureGapControlMode	220	supports stream gap control
portFeaturePatternOffsetFlexible	221	supports specification of a pattern offset based on packet component in the <i>filterPalette</i> command
portFeatureSonet	227	supports SONET
portFeatureTransceiverXenpak	231	the port supports a Xenpak interface
portFeatureXFP	232	the port supports an XFP interface.
portFeatureRepeatableRandomStreams	236	supports the ability to repeat the last set of randomly generated stream values
portFeatureGre	238	supports GRE

Feature	Value	Description
portFeatureMultiSwitchPacketDetection	243	supports the detection of multi-path switched packet loss and skip detection
portFeatureProtocolDHCP	245	supports the DHCP protocol
portFeatureUseInterfaceInStream	246	supports the ability to use an IP address from an <i>interfaceEntry</i> in a <i>stream</i>
portFeatureStackedVlan	247	supports stacked VLAN (Q in Q)
portFeatureFrequencyOffset	248	Supports the ability to alter the clock frequency. See the <i>transmitClockDeviation</i> option in this command.
portFeaturePreEmphasis	249	supports pre-emphasis specification. See the <i>preEmphasis</i> option in this command.
portFeatureTrafficMap	250	supports a traffic map
portFeatureProtocolDHCPv6	251	supports DHCPv6
portFeatureAutoDetectRx	253	supports receive side automatic instrumentation detection
portFeatureAutoDetectTx	254	supports transmit side automatic instrumentation detection
portFeatureChainUdf	255	supports chained UDFs
portFeatureStreamStartTxDelay	256	supports start stream delay
portFeatureStreamExtractor	265	supports stream extraction module features
portFeatureStreamExtractorMonitor	266	supports the monitor function of the stream extraction module
portFeatureStreamExtractorInline	267	supports the inline (receive) function of the stream extraction module
portFeatureVcat	271	supports VCAT feature
portFeatureLaps	272	supports Link Access Procedure SDH
portFeatureSplitPgId	273	supports split PGID feature.
portFeatureIncludePreambleInRxCrc	274	supports including the preamble length in the receive side CRC calculation
portFeatureTransceiverX2	276	the port supports an X2 interface
portFeatureConditionalStats	278	supports flow detective
portFeature1GEAggregate	280	supports 1GE Aggregate mode
portFeature10GEAggregate	281	supports 10GE Aggregate mode
portFeatureAdvancedStreamContBurst	282	supports continuous burst mode in advanced stream scheduler mode
portFeatureDaSa2	283	supports destination address and source address generation
portFeatureRxFilters	284	supports Rx filters
portFeatureUdfBitSize	285	supports bit-sized UDF
portFeatureSequenceNumberUdf	286	supports sequence number UDF
portFeaturePRBS	287	specifies whether this port is capable, valid, or active for tx/rx of PRBS packets. Active = the port is in PRBS mode.
portFeatureAdjustableRate	288	supports adjust rate (in streams)
portFeatureSuspendResume	289	supports the suspend/resume Tx feature
portFeatureIntrinsicLatency	290	supports intrinsic latency adjustment

Feature	Value	Description
portFeatureClearSelectedPGIDStats	293	supports clearing of selected PGID stats
portFeatureMACSec	294	supports MAC Sec Tx/Rx
portFeatureTransceiver10GBaseT	297	supports Transceiver 10G BaseT interface
portFeatureEthernetOAM	299	supports OAM port config/stream config
portFeatureDoNotApplyFrameCRC	300	port feature does not support Frame CRC application
portFeatureAdjustableFrameSize	312	supports changing frame size on the fly
portFeatureL2TP	315	adds support for checksum calculation for the inner L3/L4 (inner IP and TCP/UDP) protocols carried over L2TP
portFeatureFloatingTimestampAndDataIntegrity	317	adds timestamp as part of floating instrumentation header, and addresses similar issue in Data Integrity checking
portFeatureDualClocks	319	supports both LAN and WAN clocking concurrently
portFeatureDataCenterMode	322	enables Data Center Mode where FCoE is active; supports priority flow control (PFC) mapping
portFeatureTcpIPv4ChecksumOverride	324	supports TCP IPv4 checksum override
portFeaturePtp	325	supports IEEE1588v2 (PTP - Precision Time Protocol) 2-step only
portFeatureDataLanes	331	supports lane skew, mapping, stats
portFeature100GigEthernet	335	supports 100GE LSM XMV module
portFeature40GigEthernet	336	supports 40GE LSM XMV module
portFeatureSfpPlus	337	supports SFP+ transceiver
portFeatureDelayVariation	339	supports delay variation/jitter measurement
portFeatureMisdirectedPacket	341	supports misdirected packet count
portFeatureRateMonitoring	342	supports monitoring convergence times and service interruption
portFeatureIncrFrameBurstOverride	343	supports packet burst override in incrementing frame mode
portFeatureTransparentDynamicRateChange	345	supports transparent dynamic rate change
portFeatureLastBitTimeStamp	346	supports store and forward latency
portFeatureChecksumErrorStatsPerPGID	354	supports per-flow error statistics
portFeaturePcsLaneErrorGeneration	356	supports PCS lane error generation
portFeatureBertList	365	supports BERT in 40GE and 100GE cards
portFeatureL7Mode	370	supports L7 operation mode in NGY
portFeaturePFC	374	supports priority flow control
portFeatureUdfLinkedListMode	137	supports UDF linked list mode
portFeatureWanIFSStretch	379	supports IFS Stretch feature in WAN mode
portFeaturePCPUFlowControl	376	supports PCPU Flow Control

Feature	Value	Description
portFeatureHWIPsec	377	supports HW-IPsec
portFeaturePacketStreamsCoarse (VM only)	407	supports packet streams with less precision and cpu utilization
portFeatureAdvancedScheduler Coarse (VM only)	408	supports advance stream schedule operation with less precision and cpu utilization
portFeature1588TimeStamp	412	supports IEEE1588v2 (PTP - Precision Time Protocol)
portFeaturePFCPauseResponse Delay	413	supports the ability to increase the number of frames that is sent when a pause frame is received
portFeatureMultinicPerOS	414	supports multiple NIC per IxOS setup
portFeatureKillBitMode	418	supports Kill Bit mode statistic feature
portFeatureDynamicBackground Update	419	supports dynamic background update
portFeatureEndOfFrameTimestampAndDI		supports end of frame timestamp
portFeatureVlan0x9300	433	supports VLAN 0x9300 option
portFeatureSequenceAdv Tracking	434	supports Advanced Sequence tracking
portFeatureTransceiverCfpQsfp	429	supports CFP-QSFP transceiver
portFeatureTransceiverHse40G Qsfp	437	supports HSE 40G QSFP transceiver
portFeaturePacketLength Insertion		supports ability to insert the length of the packet in the packet
portFeatureImpairment		supports feature impairment
portFeatureDataCenter2Priority		supports Data Center 2 priority traffic mapping
portFeatureDataCenter4Priority		supports Data Center 4 priority traffic mapping
portFeatureDataCenter8Priority		supports Data Center 8 priority traffic mapping
portFeatureLinearCoefficientUdf		supports linear coefficient UDF
portFeatureTripleNestedUdf		supports triple nested UDF
portFeatureReArmFirstTimeStamp		supports first timestamp
portFeatureRestartStream		supports restart stream
portFeatureSimulateTxCable Disconnect		supports simulation of TX cable disconnection

port reset *chassisID cardID portID*

Deletes all streams from a port. Current configuration is not affected. Note: In order for **port** reset to take effect, **stream write** or **ixWriteConfigToHardware** commands should be used to commit the changes to hardware. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

port resetStreamProtocolStack *chasID cardID portID*

Sets the factory default values for all configuration options on a particular port's stream protocol stack. When this command runs, all protocols on the port are reset to their factory default state.

If the protocol stack is successfully reset, TCL_OK is returned, else TCL_ERROR is returned.

A TCL_ERROR is returned when:

- The arguments provided do not resolve to a known port known.
- The arguments provided do resolve to a known port, but that port is owned by another user.

port restartAutoNegotiation *chasID cardID portID*

Causes auto-negotiation of duplex and speed to be restarted on the indicated port.

port set *chasID cardID portID*

Sets the configuration of the port in IxHAL with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **port config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- Insufficient memory to add data

port setDefault

Sets to IxTclHal default values for all configuration options.

port setFactoryDefaults *chasID cardID portID*

Sets the factory defaults to the port. The factory defaults vary depending on the particular port type. The following two tables lists the factory defaults associated with all current board types. For ports which support streams, one default stream is written. The mode of dual PHY ports is set back to its default state. Options not mentioned in the table have a constant value as shown in the STANDARD OPTIONS section above.

Table 2: Port Type

Port Type	advertise1000FullDuplex	advertise100FullDuplex	advertise100HalfDuplex	advertise10FullDuplex	advertise10HalfDuplex	advertiseAbilities	autonegotiate	duplex
All 10/100Mbps	true	true	true	true	true	false	true	full
All 100Mbps	false	true	false	false	false	false	false	half
1000 SFPS4	false	false	false	false	false	false	false	full
All other Gigabit	false	true	true	true	true	false	true	full
All OC12c/OC3c	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
OC48 POS and POS/BERT	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
OC48 BERT	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 2: Port Type

Port Type	advertise1000 FullDuplex	advertise100 FullDuplex	advertise100 HalfDuplex	advertise10 FullDuplex	advertise10 HalfDuplex	advertise Abilities	autonegotiate	duplex
Unframed BERT	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
OC192 POS and POS/ BERT	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
OC192 BERT	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
OC192 POS/10GEWAN and POS/BERT/ 10GEWAN	false	true	true	true	true	false	false	full
All ATM	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
10GE (non-BERT)	false	true	true	true	true	false	false	full
10GE BERT	false	true	true	true	true	false	true	full
40Gig BERT Unframed	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 3: Port Type Per Flow

Port Type	flowControl	negotiateMasterSlave	portMode	receiveMode	speed	transmitMode
All 10/100Mbps	false	true	N/A	capture	max	streams
All 100Mbps	false	false	N/A	capture	100	streams
1000 SFPs4	false	false	N/A	capture	1000	streams
All other Gigabit	false	false	N/A	capture	1000	streams
All OC12c/OC3c	N/A	N/A	N/A	capture	622	streams
OC48 POS and POS/ BERT	N/A	N/A	N/A	capture	2488	streams
OC48 BERT	N/A	N/A	N/A	bert	2488	bert
Unframed BERT	N/A	N/A	N/A	bert	155	bert
OC192 POS and POS/ BERT	N/A	N/A	N/A	capture	9953	streams
OC192 BERT	N/A	N/A	N/A	bert	9953	bert
OC192 POS/10GEWAN and POS/BERT/ 10GEWAN	true	false	Ethernet Framing	capture	9953	streams
All ATM	N/A	N/A	N/A	capture	622	streams
10GE (non-BERT)	true	false	N/A	capture	10000	streams
10GE WAN (non-BERT)	true	false	N/A	capture	9294	streams

Table 3: Port Type Per Flow

Port Type	flowControl	negotiateMasterSlave	portMode	receiveMode	speed	transmitMode
10GE BERT	N/A	N/A	N/A	bert	10000	bert
10GE BERT/WAN	N/A	N/A	N/A	bert	9294	bert
40Gig BERT Unframed	N/A	N/A	N/A	bert	40000	bert

Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

port setModeDefaults *chasID cardID portID*

Sets the factory default values for all configuration options on a particular port for the current setting of *portMode*. The *portMode* option is not changed. See the tables contained in the description of the *setMode* sub-command for a listing of the default values.

port setPhyMode *phyMode chasID cardID portID*

For cards which support both Copper and Fiber PHY modes, this command sets the current PHY mode.

Option	Value	Usage
portPhyModeCopper	0	Copper
portPhyModeFiber	1	Fiber

Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

port setReceiveMode *receiveMode chasID cardID portID*

Sets the receive mode on the port. See the description for *receiveMode* in the STANDARD OPTIONS section of this command for the values of *receiveMode*. Return codes are:

Code	Usage
0	(TCL_OK) The command succeeded and a write to hardware is needed, either with <i>port write</i> or ixWritePortsToHardware .
200	(ixTcl_noWriteRequired) No write is needed to set the mode, because the port is already in that mode.
101	(ixTcl_unsupportedFeature) This port type will not support the requested receive mode.
100	(ixTcl_notAvail) This port is owned by another user.

port setTransmitMode *transmitMode chasID cardID portID*

Sets the transmission mode on the port. See the description for *transmitMode* in the STANDARD OPTIONS section of this command for the values of *transmitMode*. See the return codes in *port setReceivedMode*.

port write *chasID cardID portID*

Writes or commits the changes in IxHAL to hardware for the port. Before using this command, use the *port set* command to configure the port related parameters (speed, duplex mode, autonegotiation, flow control, loopback, rxTxMode, and ignoreLink) in IxHAL. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Network error between the client and chassis

DEPRECATED COMMANDS**port getInterface *chasID cardID portID***

Gets the interface type of the port. **port setparm *chasID cardID portID***

Modify the configuration options of the port on a specific card and chassis. It is similar to the **port config *option value*** command but allows a single *option* to be set in IxTclHAL on a particular port.

port writeReceiveMode *chasID cardID portID*

Sets up the hardware to capture or packet group modes for this port.

Note: OBSOLETE. This command is the same as write.

port writeTransmitMode *chasID cardID portID*

Sets up the hardware to packet streams or packet flow mode for this port. Note: OBSOLETE. This command is the same as write.

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
```

```
set chas [ixGetChassisID $host]

# Define all of the features by number and name
set pfValid [list \
    $::portFeatureQos                                "Qos" \
    $::portFeaturePacketFlows                         "Packet Flows" \
    $::portFeatureUdfOddOffset                       "UDF odd offset" \
    $::portFeatureRxPacketGroups                     "Rx packet groups" \
    $::portFeatureRxSequenceChecking                 "Rx sequence checking" \
    \
    $::portFeatureRxDataIntegrity                  "Rx data integrity" \
    $::portFeatureRxRoundTripFlows                 "Rx round trip flows" \
    \
    $::portFeatureGigGMiiAutoDisable \
    disable"                                         "Gig GMII auto" \
    $::portFeatureMultipleDLCIs                   "Multiple DLCIs" \
    $::portFeatureForcedCollisions                "Forced Collisions" \
    $::portFeatureTxDataIntegrity                 "Tx Data Integrity" \
    file" \
    $::portFeatureSrp                                "Spacial reuse" \
    protocol" \
    $::portFeaturePos                               "POS" \
    $::portFeatureBert                             "Bert" \
    $::portFeature10GigWan                         "10 Gigabit WAN" \
    $::portFeatureUdfOverlap                        "UDF Overlap" \
    $::portFeatureUdfCascade                        "UDF Cascade" \
    $::portFeatureRxSequenceCheckingPerPGID       "Rx Seq Checking per PGID" \
    $::portFeaturePacketStreams                    "Packet Streams" \
    $::portFeatureAdvancedScheduler                "Advanced Scheduler" \
    $::portFeatureProtocols                        "Protocols" \
    $::portFeatureProtocolARP                     "Protocol: ARP" \
    $::portFeatureProtocolPING                   "Protocol: PING" \
    $::portFeatureBitMask                          "Bit Mask" \
    $::portFeatureSonetErrorInsertionList        "Sonet error" \
    insertion list" \
    $::portFeatureBertErrorGeneration            "BERT error" \
    generation" \
    $::portFeatureLocalCPU                         "Local CPU" \
    \
    $::portFeatureIxRouter                         "IxRouter" \
    $::portFeatureIxWeb                            "IxWeb" \
    $::portFeature10GigLan                         "10G LAN" \
    \
    $::portFeatureVsr                             "VSR" \
    $::portFeatureSplitUdfs                      "Split UDF" \
    \
    $::portFeatureTxDuration                     "Transmit Duration" \
    \
    Sessions" \
    $::portFeatureRxFirstTimeStampStamp           "Received First Time" \
    $::portFeatureRxStreamTriggerTrigger          "Received Stream" \
    $::portFeatureRxChecksumErrorsErrors          "Received Checksum" \
    $::portFeatureOddPreamble                    "Odd Preamble" \
    $::portFeaturePacketGapTimeUnitsUnits         "Packet Gap Time" \
    Units" \
    $::portFeatureRoutingProtocols               "Routing Protocols" \
    $::portFeatureModifiablePreamble            "Modifiable Preamble" \
    ]
```

```

$::portFeatureIgnorePGIDSignature           "Ignore PGID
Signature" \
$::portFeatureBertUnframed                 "Unframed BERT" \
$::portFeatureXaui                         "XAUI" \
$::portFeatureBertChannelized              "Channelized BERT" \
$::portFeatureLdp                          "Protocol: LDP" \
$::portFeatureUdf5                        "UDF5" \
$::portFeatureTxDccStreams                "DCC Streams" \
$::portFeatureTxDccAdvancedScheduler     "DCC Advanced
Scheduler" \
$::portFeatureTxDccFlowsSpeStreams        "DCC Flows SPE
Streams" \
$::portFeatureTxDccFlowsSpeAdvancedScheduler "DCC Flows SPE
Adv Scheduler" \
$::portFeatureRxDcc                       "DCC Receive" \
$::portFeatureDccProperties               "DCC Properties" \
$::portFeatureProtocolL2VPN               "Protocol: L2VPN" \
$::portFeatureProtocolL3VPN               "Protocol: L3VPN" \
$::portFeatureProtocolRIPng              "Protocol: RIPng" \
$::portFeatureSrpFullFeatured            "SRP Full Featured" \
$::portFeatureUdfExtension1              "UDF Extension 1" \
$::portFeatureTxFrequencyDeviation      "Transmit Freq
Deviation" \
$::portFeatureUdfTableMode                "UDF Value List Mode"
\
$::portFeatureCapture                     "Capture" \
$::portFeaturePauseControl                "Pause Control" \
$::portFeatureCJPAT                      "CJPAT" \
$::portFeatureCRPAT                      "CRPAT" \
$::portFeatureProtocolIGMP               "Protocol: IGMP" \
$::portFeatureAtm                        "ATM" \
$::portFeatureRpr                        "RPR" \
$::portFeatureLinkFault                  "Link Fault Signaling"
\
$::portFeatureProtocolMLD                "Protocol: MLD" \
$::portFeatureProtocolPIMSM              "Protocol: PIMSM" \
$::portFeatureProtocolOSPFv3             "Protocol: OSPFv3" \
$::portFeatureIPv6NeighborDiscovery     "IPv6 Neighbor
Discovery" \
$::portFeatureProtocolBGPv6              "Protocol: BGPv6" \
$::portFeatureProtocolISISv6             "Protocol: ISISv6" \
$::portFeatureFlexibleTimestamp         "Flexible Time Stamp"
\
$::portFeatureProtocolOffset             "Protocol Offset" \
$::portFeatureRandomGap                 "Random Gap" \
$::portFeatureLayer7Only                "Layer 7 Only" \
$::portFeatureUniphy                   "UNIPHY" \
$::portFeatureUdfIPv4Mode              "UDF IPv4 Mode" \
$::portFeatureRandomFrameSizeWeightedPair "Random Frame Size
Weighted Pair" \
$::portFeatureRxWidePacketGroups       "Receive Wide Packet
Groups" \
$::portFeatureDualPhyMode              "Dual PHY Mode" \
$::portFeatureAtmPos                  "ATM POS" \
$::portFeatureFec                     "FEC" \
$::portFeatureAtmPatternMatcher       "ATM Pattern Matcher"
\
$::portFeatureGfp                      "GFP" \
$::portFeatureCiscoCDL                "Cisco CDL" \
$::portFeatureRxLatencyBin             "Receive Latency Bins"
\
$::portFeatureRxTimeBin                "Receive Time Bins" \
$::portFeaturePreambleView             "View Preamble" \

```

```

$::portFeaturePreambleCapture           "Capture Preamble" \
$::portFeatureCDLErrorTrigger          "CDL Error Trigger" \
$::portFeatureSimulateCableDisconnect "Simulate Cable"
Disconnect" \
$::portFeatureXfp                      "XFP" \
$::portFeatureTableUdf                 "Table UDF" \
$::portFeatureOc192                     "OC192" \
$::portFeaturePerStreamTxStats         "Per-Stream Transmit
Stats" \
$::portFeatureLasi                     "LASI" \
$::portFeaturePowerOverEthernet        "PoE" \
$::portFeatureGapControlMode          "Gap Control Mode" \
$::portFeaturePatternOffsetFlexible   "Flexible Pattern
Offset" \
$::portFeatureSonet                   "SONET" \
$::portFeatureRepeatableRandomStreams "Repeatable Random
Streams" \
$::portFeatureGre                     "GRE" \
$::portFeatureMultiSwitchPacketDetection "Multi-Path Switched
Packet Detection" \
$::portFeatureProtocolDHCP            "Protocol: DHCP" \
$::portFeatureUseInterfaceInStream    "Use Interfaces in
Streams" \
$::portFeatureStackedVlan             "Stacked VLAN" \
$::portFeatureFrequencyOffset         "Frequency Offset" \
$::portFeaturePreEmphasis            "Pre-Emphasis" \
]

# Define all of the features by number and name
set pfActive [list \
$::portFeatureRxPacketGroups          "Rx packet groups" \
$::portFeatureRxDataIntegrity         "Rx data integrity" \
$::portFeatureRxRoundTripFlows        "Rx round trip flows"
\
$::portFeaturePos                     "POS" \
$::portFeatureBert                   "BERT" \
$::portFeature10GigWan                "10 Gigabit WAN" \
$::portFeatureBertErrorGeneration    "BERT error
generation" \
]

# printOptions - get standard options for a port and print them
proc printOptions {chash card port} {
    port get $chash $card $port
    set portType [port cget -type]
    set portName [port cget -typeName]
    set name [port cget -name]
    set owner [port cget -owner]
    set linkState [port cget -linkState]
    set rateMode [port cget -rateMode]
    set loopback [port cget -loopback]
    set flowControl [port cget -flowControl]
    set portMode [port cget -portMode]
    ixPuts "Port: $name, type $portName ($portType)"
    ixPuts "\towner $owner, linkState $linkState, rateMode
$rateMode"
    ixPuts "\tloopback $loopback, flowControl $flowControl,
portMode $portMode"
}

# Print the values of all of the 'valid' features
proc printValid {chash card port} {
    global pfValid;

```

```

array set portValidFeatures $pfValid

foreach i [lsort -integer [array names portValidFeatures]] {
    if {[port isValidFeature $chassis $card $port $i] == 0} {
        ixPuts -nonewline "No "
    } else {
        ixPuts -nonewline "Yes "
    }
    ixPuts $portValidFeatures($i)
}
}

# Print the values of all of the 'active' features
proc printActive {chassis card port} {
    global pfActive;
    array set portActiveFeatures $pfActive;

    foreach i [lsort -integer [array names portActiveFeatures]] {
        if {[port isActiveFeature $chassis $card $port $i] == 0} {
            ixPuts -nonewline "No "
        } else {
            ixPuts -nonewline "Yes "
        }
        ixPuts $portActiveFeatures($i)
    }
}

# Get the chassis' number of cards
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
chassis getFromID $chassis
set ncards [chassis cget -maxCardCount]
ixPuts "Chassis $chassis, $ncards cards"

# Go through each of the ports
for {set i 1} {$i <= $ncards} {incr i} {
    # Check for missing card
    if {[card get $chassis $i] != 0} {
        continue
    }
    ixPuts "\n-----"
    set portList [list [list $chassis $i 1]]
    # Take ownership of the ports we'll use
    if [ixTakeOwnership $portList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Get the port's options
    port get $chassis $i 1

    # Get the type of the card as a number and name
    set portType [port cget -type]
    set portName [port cget -typeName]
    set cardType [card cget -type]
    ixPuts "Type $portName ($portType) -- card $i (type
$cardType)"

    # Set the port to its defaults
}

```

```
port setDefault
# If it's a BERT module, need to set the transmit and receive
modes to BERT
if {$portType == $::portOc48Bert} {
    port config -transmitMode portTxModeBert
    port config -receiveMode portRxModeBert
}

# Give the port a name
port config -name "$i:1"

# Set the features to ixTclHal
set result [port set $chassis $i 1]
# Check for valid result
if {$result != 0} {
    ixPuts "Set returns $result"
    continue
}

# Write the features to the hardware
ixWriteConfigToHardware portList

# Print the standard options for the card
printOptions $chassis $i 1

# Print the valid features for the card
ixPuts "\nValid Features for Port"
ixPuts "-----"
printValid $chassis $i 1

# Set some values for the 10/100 cards:
# Rx:capture, Tx:packet flows w/ image file
# Autonegotiate 100 Half or 100 Full duplex
if {$portType == $::port10100BaseTX} {
    port config -receiveMode portCapture
    port config -transmitMode portTxPacketFlows
    port config -usePacketFlowImageFile 1
    port config -packetFlowFileName "flow10100.txt"
    port config -autonegotiate true
    port config -advertise10HalfDuplex false
    port config -advertise10FullDuplex false
    port config -advertise100HalfDuplex true
    port config -advertise100FullDuplex true
    port config -speed 100
}

# Set some values for OC48c POS cards
if {$portType == $::portPacketOverSonet} {
    port config -receiveMode portPacketGroup
}
# Set the values
port set $chassis $i 1
ixWriteConfigToHardware portList

# Check on what features are active now
ixPuts "\nActive Features for Port"
ixPuts "-----"
printActive $chassis $i 1
ixClearOwnership $portList
}

# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
```

```
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

card, filter, filterPallette, portGroup, stream, packetGroup

NAME - portCpu

portCpu - control a port's CPU.

SYNOPSIS	portCpu <i>sub-command options</i>
DESCRIPTION	<p>This command allows to control the CPU associated with many Ixia load modules. The <i>port</i> command's <i>isValidFeature</i> sub-command may be used to determine if a given port has a CPU. Use the following sequence:</p> <pre>if [port isValidFeature \$chassis \$card \$port portFeatureLocalCPU] { ... port has a CPU ... }</pre> <p>The only sub-command currently available is the <i>reset</i> command, which causes the port to reboot its operating system and return to its initial state. Any optional loaded packages are removed.</p>
<hr/>	
STANDARD OPTIONS	
memory	<i>Read-only</i> . The amount of memory, expressed in Mbytes, associated with the CPU on the port.
<hr/>	
COMMANDS	<p>The portCpu command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.</p>
portCpu cget option	<p>Returns the current value of the configuration option given by <i>option</i>. <i>Option</i> may have any of the values accepted by the portCpu command.</p>
portCpu get chassisID cardID portID	<p>Gets the current configuration of the port CPU for the indicated port. Call this command before calling port cget option value to get the value of the configuration option. Specific errors are:</p> <ul style="list-style-type: none"> • No connection to a chassis • Invalid port number
portCpu reset chassisID cardID portID	<p>Resets the CPU on the indicated port. Specific errors are:</p> <ul style="list-style-type: none"> • No connection to a chassis • The port is owned by another user • The port does not have a local CPU
<hr/>	
EXAMPLES	<pre>package require IxTclHal set host localhost set username user # Assume card 1 is a card that has a CPU set card 1</pre>

```
set port 1

if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

set chas [ixGetChassisID $host]

if {[port isValidFeature $chas $card 1 portFeatureLocalCPU] == 0}
{
    ixPuts "$chas:$card does not have a local CPU"
    return 1
}

if [portCpu reset $chas $card $port] {
    ixPuts $::ixErrorInfo
    return 1
}
ixPuts "$chas:$card:$port has been reset"

if [portCpu get $chas $card $port] {
    ixPuts $::ixErrorInfo
    return 1
}
ixPuts "Port $chas:$card:$port has [portCpu cget -memory] MB of
memory"
```

SEE ALSO

port

NAME - portGroup

portGroup - sets up a group of ports.

SYNOPSIS	portGroup <i>sub-command options</i>
DESCRIPTION	This command allows to set up an autonomous group of ports on which to perform an action or command, such as take ownership, start transmit, capture, or clearing statistics, to name a few. A port group must be created and the desired ports (or port) added to it to execute the selected action or command. When the port group is no longer needed, it should be destroyed.
STANDARD OPTIONS	
lastTimeStamp	<i>Read-only. 64-bit value.</i> The relative time of transmit for all the ports in the port group.
COMMANDS	The portGroup command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
portGroup add <i>groupID chasID cardID portID</i>	Adds this port to a group with ID <i>groupID</i> . Specific errors are: <ul style="list-style-type: none"> • No connection to a chassis • The <i>groupID</i> port group does not exist
portGroup canUse <i>groupID</i>	Verifies whether all the ports in this group can be used by the current logged in user. Specific errors are: <ul style="list-style-type: none"> • No connection to a chassis • The <i>groupID</i> port group does not exist
portGroup cget <i>option</i>	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the portGroup command.
portGroup clearScheduledTransmitTime <i>groupID</i>	Clears the scheduled transmit time associated with a group of ports. See <i>setScheduledTransmitTime</i> . Specific errors are: <ul style="list-style-type: none"> • No connection to a chassis • The <i>groupID</i> port group does not exist
portGroup config <i>option value</i>	Modify the configuration options of all the ports. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for portGroup. (There are currently no configurable options for portGroup and therefore no use for this command).

portGroup create *groupID*

Creates a port group and assigns it the ID *groupID*. Specific errors are:

- The groupID port group already exists

portGroup del *groupID chassisID cardID portID*

Deletes this port from the group with ID *groupID*. Specific errors are:

- No connection to a chassis
- The groupID port group does not exist

portGroup destroy *groupID*

Destroys the port group with ID *groupID*. Specific errors are:

- The groupID port group does not exist

portGroup setCommand *groupID cmd*

Performs the action or command *cmd* specified on all ports in the group with ID *groupID*. Note that some of the command values previously listed in this table have been moved to the *IxRouter Tcl Development Guide*. *cmd* may be one of the following:

Option	Value	Usage
<i>startTransmit</i>	7	start transmission simultaneously
<i>stopTransmit</i>	8	stop transmission simultaneously
<i>startCapture</i>	9	start capturing packets that meet the specified filter criteria
<i>stopCapture</i>	10	stop capturing simultaneously
<i>resetStatistics</i>	13	clear all statistic counters
<i>pauseTransmit</i>	15	pause transmission
<i>stepTransmit</i>	16	single step the transmit one packet at a time
<i>transmitPing</i>	17	transmit a ping on all of the ports in the port group
<i>asr5Transmit</i>	18	not yet implemented
<i>clearTimeStamp</i>	19	clear all time stamps to synchronize the time stamps throughout the chassis chain.
<i>restartAutoNegotiate</i>	22	restarts autonegotiation
<i>downloadFPGA</i>	27	downloads a new FPGA to the ports in the port group
<i>collisionStart</i>	28	start collision generation
<i>collisionStop</i>	29	stop collision generation
<i>transmitArpRequest</i>	30	sends ARP requests as configured in ipAddressTable commands.
<i>startLatency</i>	31	starts real-time latency analysis and collects minimum, maximum and average latency values for every incoming frame in a packet group. Ensure to clear timestamps on all send and receive ports before starting latency measurements.
<i>stopLatency</i>	32	stops real-time latency analysis
<i>clearLatency</i>	33	clears all real-time latency values
<i>takeOwnership</i>	40	if available, take ownership of these ports
<i>takeOwnershipForced</i>	41	forcefully take ownership of these ports overriding the current owner's rights

Option	Value	Usage
<i>clearOwnership</i>	42	clear ownership of owned ports
<i>clearOwnershipForced</i>	43	forcefully clear ownership of ports overriding the current owner's rights
<i>clearArpTable</i>	48	clear ports' arp tables
<i>staggeredStartTransmit</i>	51	start transmit in sequence
<i>resetSequenceIndex</i>	62	resets the sequence number used in sequence number checking operations for all ports in the portGroup
<i>rebootLocalCPU</i>	84	causes the CPU to reboot, if the port uses a CPU
<i>clearPerStreamTxStats</i>	120	clears the per stream statistics
<i>loadPoEPulse</i>	121	for PoE load modules, causes a power pulse
<i>armPoeTrigger</i>	123	arm the triggers for <i>poeSignalAcquisition</i>
<i>abortPoeArm</i>	124	abort the triggers for <i>poeSignalAcquisition</i>
<i>startAtmOamTx</i>	125	starts transmission of the ATM OAM messages
<i>stopAtmOamTx</i>	126	stops transmission of the ATM OAM messages
<i>simulatePhysicalInterfaceDown</i>	128	sets the port to simulate a downed interface
<i>simulatePhysicalInterfaceUp</i>	129	reenables the interface after setting the port to simulate a downed interface.
<i>clearPrbsCapture</i>	139	Clears the PRBS capture buffer
<i>startTxRxSyncStats</i>	146	Starts collecting Tx/Rx Sync stats
<i>stopTxRxSyncStats</i>	147	Stops collecting Tx/Rx Sync stats
<i>clearThresholdTime</i>	154	Clears the threshold timestamps associated with a group of ports.
<i>clearPcsLaneStats</i>	155	Clears PCS Lane stats

Specific errors are:

- No connection to a chassis
- One or more ports in the port group are being used by another user
- One or more ports in the port group are invalid
- Network error between the client and chassis

portGroup **setDefault**

Sets to IxTclHal default values for all configuration options.

portGroup **setScheduledTransmitTime** *groupID time*

This feature only applies to ports which support the *portFeatureScheduledTxDuration* feature (see *port* is *isValidFeature*). This sub-command sets the transmit time duration associated with the group of ports. *time* is expressed in seconds. When a scheduled transmit time is set, and a *portGroup setCommand <group> startTransmit* is issued, the ports in the port group transmits until their streams are exhausted or the specified *time* has elapsed, whichever comes first. This value may be cleared with the *clearScheduledTransmitTime* sub-command to this command. Specific errors are:

- No connection to a chassis
- The *groupID* port group does not exist
- Invalid *time* value.

portGroup startPrbsCapture *portlist*

Starts PRBS capture on specified ports. This command also starts *packetGroup* stat collection.

portGroup stopPrbsCapture *portlist*

Stops PRBS capture on specified ports. This command also stops *packetGroup* stat collection

portGroup write *groupID* [*writeProtocolServer*]

Commits port properties information such as speed, duplex mode, and autonegotiation in hardware. If *writeProtocolServer* is true, then the protocol server is stopped and all applicable objects written to it. Otherwise, the protocol server is not affected. Specific errors are:

- No connection to a chassis
- The port group specified by *groupID* hasn't been created
- One or more ports in the port group are being used by another user
- Network error between the client and chassis

portGroup writeConfig *groupID* [*writeProtocolServer*]

Configures streams, filter and capture parameters of all ports in the group except the port properties such as speed, duplex mode, and autonegotiation. If *writeProtocolServer* is true, then the protocol server is stopped and all applicable objects written to it. Otherwise, the protocol server is not affected. Specific errors are:

- No connection to a chassis
- The port group specified by *groupID* hasn't been created
- One or more ports in the port group are being used by another user
- Network error between the client and chassis

DEPRECATED COMMANDS

portGroup get *groupID objectID*

Gets the type of object designated by *objectID* for a list of ports. The only defined value for *objectID* is *usbConfig (0)*, which must be applied to USB configured ports. Specific errors are:

- Invalid *objectID*
- The *groupID* port group does not exist

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
```

```
}
```

```
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assume that there's a four port 10/100 TX card in this slot
# with port 1 looped to port 2 and 3 to 4
set card 1
set portList [list [list $chas $card 1] \
    [list $chas $card 2] \
    [list $chas $card 3] \
    [list $chas $card 4]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
# Port group to be used
set group 1234
set txGroup 13
set rxGroup 24

portGroup create $group
portGroup add $group $chas $card 1
portGroup add $group $chas $card 2
portGroup add $group $chas $card 3
portGroup add $group $chas $card 4

portGroup create $txGroup
portGroup add $txGroup $chas $card 1
portGroup add $txGroup $chas $card 3

portGroup create $rxGroup
portGroup add $rxGroup $chas $card 2
portGroup add $rxGroup $chas $card 4

if {[portGroup canUse $group] != 0} {
    ixPuts "Can't use card $card ports 1-4"
    break
}
portGroup setCommand $group takeOwnership

# ... insert port setup here. This example assumes the defaults
portGroup write $group

portGroup setCommand $rxGroup resetStatistics
portGroup setCommand $rxGroup startCapture
portGroup setCommand $txGroup startTransmit
after 5000
portGroup setCommand $txGroup stopTransmit
portGroup setCommand $rxGroup stopCapture
```

```
portGroup setCommand $group clearOwnership
portGroup destroy $group
portGroup destroy $rxGroup
portGroup destroy $txGroup

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

port, prbsCapture

NAME - ppp

ppp - configure Point-To-Point Protocol parameters

SYNOPSIS	ppp <i>sub-command options</i>
DESCRIPTION	This command is used to configure PPP parameters on OC-* interfaces for Packet over Sonet ports.
STANDARD OPTIONS	
activeNegotiation true/false	Activate Negotiation process. (<i>default = true</i>)
configurationRetries	Maximum number of configuration requests to send before starting termination process. (<i>default = 9</i>)
enable true / false	Enable PPP negotiations. (<i>default = false</i>)
enableAccmNegotiation true / false	Enables ACCM (Asynchronous Control Character Mask). (<i>default = false</i>)
enableIp true / false	Enables the IPV6 Network Control protocol. (<i>default = true</i>)
enableIpV6 true / false	Enables the IP Network Control protocol. The port's desired IP address is held in the <i>localIPAddress</i> option. (<i>default = false</i>)
enableLqm true / false	Enables the LQM (Line Quality Monitoring Protocol). The LQM reporting interval is controlled by the <i>lqmReportInterval</i> option. (<i>default = false</i>)
enableMpls true / false	Enables the MPLS Network Control protocol. (<i>default = true</i>)
enableOsi true / false	Enables the OSI Network Control protocol. The port's desired transmitted and received alignments are held in the <i>rxAlignment</i> and <i>txAlignment</i> options. (<i>default = true</i>)
localIPAddress	Local port's IP address. (<i>default = 0.0.0.1</i>)
localIpV6Id	When the value of <i>localIpV6IdType</i> is <i>pppIpV6IdTypeIpV6</i> , this value is used to generate an Interface ID. (<i>default = {00 00 00 00 00 00 00 00}</i>)
localIpV6IdType	The type of Interface Identifier, which is a configuration option sent in the configuration request packet. The choices are:

Option	Value	Usage
<i>pppIpV6IdTypeLastNegotiated</i>	4	The last Interface Identifier that was negotiated for this link is used.
<i>pppIpV6IdTypeMacBased</i>	1	The Interface Identifier is derived from the MAC address in <i>localIpV6MacBased lid</i> .
<i>pppIpV6IdTypeIpV6</i>	2	The Interface Identifier is the 64-bit EUI-64 identifier found in <i>localIpV6lid</i> .

Option	Value	Usage
ppplpV6IdTypeRandom	8	(default) The Interface Identifier is randomly generated.

localIpV6MacBasedIid

When the value of *localIpV6IdType* is *ppplpV6IdTypeMacBased*, this value is used to generate a globally unique Interface ID. (default = {00 00 00 00 00 00 00 00})

localIpV6Negotiation Mode

Before the negotiation of the Interface Identifier (Iid), the node chooses a tentative Interface-Identifier. The choices are:

Option	Value	Usage
ppplpLocalNegotiationLocalMay	0	(default) The local node may use the lid mode and the lid value specified in <i>localIpV6IdType</i> , <i>localIpV6MacBasedIid</i> and <i>localIpV6lid</i> .
ppplpLocalNegotiationLocalMust	1	The local node must use the lid mode and the lid value specified in <i>localIpV6IdType</i> , <i>localIpV6MacBasedIid</i> and <i>localIpV6lid</i> .
ppplpLocalNegotiationPeerMust	2	The peer node must supply the local lid.

lqmReportInterval

The desired LQM report interval, expressed in seconds. (default = 10.0)

peerIpV6Iid

When the value of *peerIpV6IdType* is *ppplpV6IdTypeIpV6*, this IPv6 address is used to generate an Interface ID. This value must be unique on the link. (default = {00 00 00 00 00 00 00 00})

peerIpV6IdType

The type of Interface Identifier. The choices are:

Option	Value	Usage
ppplpV6IdTypeLastNegotiated	0	The last Interface Identifier that was negotiated for this link is used.
ppplpV6IdTypeMacBased	1	The Interface Identifier is derived from the MAC address in <i>peerIpV6MacBasedIid</i> .
ppplpV6IdTypeIpV6	2	The Interface Identifier is the 64-bit EUI-64 identifier found in <i>peerIpV6lid</i> .
ppplpV6IdTypeRandom	8	(default) The Interface Identifier is randomly generated.

peerIpV6MacBasedIid

When the value of *peerIpV6IdType* is *ppplpV6IdTypeMacBased*, this value is used to generate a globally unique Interface ID. This value must be unique on the link. (default = {00 00 00 00 00 00 00 00})

peerIpV6Negotiation Mode

The peer Interface Id negotiation mode. The choices are:

Option	Value	Usage
ppplpPeerNegotiationPeerMay	0	(default) The peer node may use the lid mode and the lid value specified in <i>peerIpV6IdType</i> , <i>peerIpV6MacBasedIid</i> and <i>peerIpV6lid</i> .
ppplpPeerNegotiationPeerMust	1	The peer node must use the lid mode and the lid value specified in <i>peerIpV6IdType</i> , <i>peerIpV6MacBasedIid</i> and <i>peerIpV6lid</i> .

Option	Value	Usage
pppIpPeerNegotiationLocalMust	2	The local node must supply the peer lid.

retryTimeout

Time, in seconds, to wait between configuration and termination retries. (*default = 3*)

rxAlignment

The desired OSI receive byte alignment (within a 4-byte word), expressed as a byte position from 0 to 3. (*default = 0*)

rxMaxReceiveUnit

Maximum frame size in receive direction. (*default = 65535*)

terminationRetries

Max # of termination requests to send before bringing PPP down. (*default = 3*)

txAlignment

The desired OSI transmit byte alignment (within a 4-byte word), expressed as a byte position from 0 to 3. (*default = 0*)

txMaxReceiveUnit

Maximum frame size in transmit direction. (*default = 65535*)

useMagicNumber

Enable negotiation and use of magic number; used to detect looped back connection. (*default = true*)

COMMANDS

The **ppp** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ppp cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ppp** command.

ppp config option value

Modify the PPP configuration options of the port. If no *option* is specified, returns a list describing all of the available PPP options (see STANDARD OPTIONS) for port.

ppp get chasID cardID portID [circuitID]

Gets the current configuration of the PPP parameters on circuit *circuitID*, port with id *portID* on card *cardID*, chassis *chasID*. from its hardware. Call this command before calling **ppp cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is not a Packet over Sonet port.

ppp set chasID cardID portID [circuitID]

Sets the configuration of the PPP parameters in IxHAL on circuit *circuitID*, port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **ppp config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- The port is not a Packet over Sonet port.

ppp setDefault

Sets to IxTclHal default values for all configuration options.

ppp write *chasID cardID portID*

Writes the ppp config to the ppp state machine and restarts ppp autonegotiation. Writes or commits the changes in IxHAL to hardware for each port with id *portID* on card *cardID*, chassis *chasID*. Before using this command, use the *ppp set* command to configure the port related parameters in IxHAL. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- Network error between the client and chassis
- The port is not a Packet over Sonet port

Note: Not available at circuit level..

EXAMPLES

```
package require IxTclHal

# Procedure to get and print the status of a POS port
proc getState {chas card port} \
{
    # Get all of the status information
    pppStatus get $chas $card $port

    # IP related information
    set ipState      [pppStatus cget -ipState]
    set ipAddr       [pppStatus cget -localIPAddress]
    set ipPeerAddr   [pppStatus cget -peerIPAddress]

    # LQM State information
    set lqmState     [pppStatus cget -lqmQualityState]
    set lqmRxInterval [pppStatus cget -lqmReportIntervalRx]
    set lqmTxInterval [pppStatus cget -lqmReportIntervalTx]

    # MPLS state
    set mplsState    [pppStatus cget -mplsState]

    # OSI information
    set osiState     [pppStatus cget -osiState]
    set rxAlignment  [pppStatus cget -rxAlignment]
    set txAlignment  [pppStatus cget -txAlignment]

    # Magic numbers
    set magicRxState [pppStatus cget -useMagicNumberRx]
    set magicTxState [pppStatus cget -useMagicNumberTx]

    # Negotiated MRUs
    set rxMRU         [pppStatus cget -rxMaxReceiveUnit]
    set txMRU         [pppStatus cget -txMaxReceiveUnit]

    ixPuts "Port $chas:$card:$port"
    ixPuts "\tMRU:\trxMaxReceiveUnit $rxMRU, txMaxReceiveUnit
$txMRU"
```

```

        ixPuts "\tMagic:\tuseMagicNumberRx $magicRxState,
useMagicTxState $magicTxState"
        ixPuts "\tLQM:\tlqmReportIntervalRx $lqmRxInterval,
lqmReportIntervalTx $lqmTxInterval"
        ixPuts "\tIP:\tstate $ipState, localIpAddress $ipAddr,
peerIpAddress $ipPeerAddr"
        ixPuts "\tOSI:\tstate $osiState, rxAlignment $rxAlignment,
txAlignment $txAlignment"
        ixPuts "\tMPLS:\tstate $mplsState"
    }

# Symbolic definition of the PPP related port link states
# Not all states are necessarily defined
set pppState($::pppOff)                      "pppOff\t"
set pppState($::pppUp)                        "pppUp\t"
set pppState($::pppDown)                       "pppDown\t"
set pppState($::pppInit)                       "pppInit\t"
set pppState($::pppWaitForOpen)                 "pppWaitForOpen"
set pppState($::pppAutoNegotiate)               "pppAutoNegotiate"
set pppState($::pppClose)                      "pppClose"
set pppState($::pppConnect)                     "pppConnect"
set pppState($::pppRestartNegotiation)          "pppRestartNegotiation"
set pppState($::pppRestartInit)                 "pppRestartInit"
set pppState($::pppRestartWaitForOpen)           "pppRestartWaitForOpen"
set pppState($::pppRestartWaitForClose)          "pppRestartWaitForClose"
set pppState($::pppRestartFinish)                "pppRestartFinish"
set pppState($::pppClosing)                     "pppClosing"
set pppState($::pppLcpNegotiate)                "pppLcpNegotiate"
set pppState($::pppAuthenticate)                "pppAuthenticate"
set pppState($::pppNcpNegotiate)                "pppNcpNegotiate"
set pppState($::lossOfFrame)                    "lossOfFrame"

# Connect to chassis and get chassis ID
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that an OC12 card is in slot 2
# And that port 1 is directly connected to port 2
set card 2
set portList [list [list $chas $card 1] [list $chas $card 2]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use

```

```

if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the type of card and check if it's the correct type
set ifType [card getInterface $chassis $card]
if {$ifType != $::interfacePacketOverSonet} \
{
    ixPuts "Card $card is not an OC12c POS card"
    return 1
}
# Disable PPP negotiation and tell both ports to stop
ppp config -enable disable
ppp set $chassis $card 1
ppp set $chassis $card 2
ppp write $chassis $card 1
ppp write $chassis $card 2

# Start with a default setup, enable PPP and set auto negotiation
ppp setDefault
ppp config -enable enable
ppp config -activeNegotiation true

# Enable IP address negotiation and set our desired IP address
ppp config -enableIp enable
ppp config -localIPAddress 192.168.5.100

# Enable MPLS negotiation
ppp config -enableMpls enable

# Enable magic number negotiation
ppp config -useMagicNumber true

# Enable LQM and set the desired report interval to 1.2 seconds
ppp config -enableLqm enable
ppp config -lqmReportInterval 1.2

# Enable OSI negotiation with alignment at byte 2
ppp config -enableOsi enable
ppp config -rxAlignment 2
ppp config -txAlignment 2

# Set PPP parameters to port 1
ppp set $chassis $card 1
ppp write $chassis $card 1

# When two Ixia ports are connected directly, only one can use
recovered clock
sonet setDefault
sonet config -useRecoveredClock false
sonet set $chassis $card 1
set portList [list [list $chassis $card 1]]
ixWritePortsToHardware portList

# Change the requested address for the second port
ppp config -localIPAddress 192.168.6.100
ppp set $chassis $card 2
ppp write $chassis $card 2

# Now monitor and print the port link state until both ports show
up or a minute
# Has gone by

```

```
ixPuts "Link state monitoring"
ixPuts "Port 1\t\t\tPort 2"
ixPuts "----\t\t\t----"
for {set i 0} {$i < 60} {incr i} \
{
    after 1000
    port get $chas $card 1
    set portState1 [port cget -linkState]
    port get $chas $card 2
    set portState2 [port cget -linkState]
    ixPuts "$pppState($portState1)\t\t$pppState($portState2)"
    if {$portState1 == $::pppUp && $portState2 == $::pppUp}
    {break}
}
# If both ports went to pppUp, then get and print the state for
each
if {$portState1 == $::pppUp && $portState2 == $::pppUp} \
{
    getState $chas $card 1
    getState $chas $card 2
}

# Now wait for two received LQM reports on port 1
for {set i 0} {$i < 10} {incr i} \
{
    after 1000
    pppStatus get $chas $card 1
    set lqmRxCounter [pppStatus cget -lqmReportPacketCounterRx]
    if {$lqmRxCounter >= 2} {
        ixPuts "Received 2 LQM reports"
        break
    }
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

pppStatus

NAME - pppStatus

pppStatus - configure PPP parameters

SYNOPSIS	pppStatus <i>sub-command options</i>
-----------------	--------------------------------------

DESCRIPTION	This command gets PPP status information for Packet over Sonet ports.
--------------------	---

STANDARD OPTIONS

ipState *Read-only*. The current state of the IP Network Control Protocol negotiation.

State	Value	Usage
pppStatusDisabled	0	The IPCP protocol has been disabled and is not negotiated during the NCP phase.
pppStatusClose	1	The IPCP protocol is enabled but is currently closed. IPCP traffic is dropped.
pppStatusNegotiation	2	The IPCP protocol is currently being negotiated on the link. This state may continue indefinitely if the peer refuses to negotiate IPCP.
pppStatusOpen	3	The IPCP protocol is currently open and IPCP traffic may flow.

ipV6State *Read-only*. The current state of the IPV6 Network Control Protocol negotiation. The possible values are discussed in *ipState*.

localIPAddress *Read-only*. The negotiated local IP address for the port as a result of the IP Network Control Protocol's operation.

localIpV6Id *Read-only*. The negotiated local IPV6 Interface Id for the port as a result of the IPV6 Network Control Protocol's operation.

lqmQualityState *Read-only*. The current state of the LQM negotiation.

State	Value	Usage
pppStatusNotNegotiated	0	The LQM option has been locally disabled and is not negotiated. Any subsequent Link Quality Reports (LQR) received on the link is ignored.
pppStatusInactive	1	LQM is not running on the link and any LQRs received is ignored.
pppStatusActive	2	LQM operation was agreed to by both peers during LCP negotiation and LQM is running on the link. LQRs received on the link is pre-processed and local LQRs is generated and sent.

lqmReportIntervalRx *Read-only*. The negotiated LQM receive port interval, expressed in seconds.

lqmReportIntervalTx *Read-only*. The negotiated LQM transmit port interval, expressed in seconds.

lqmReportPacketCounterRx *Read-only*. The number of LQM report packets received since link was last established.

lqmReportPacketCounterTx

Read-only. The number of LQM report packets transmitted since link was last established.

magicNumberNegotiated

Read-only. The magic number negotiated between the local and remote hosts. (*default = 0*)

mplsState

Read-only. The current state of the MPLS Network Control Protocol negotiation.

State	Value	Usage
pppStatusDisabled	0	The MPLS NCP protocol has been disabled and is not negotiated during the NCP phase.
pppStatusClose	1	The MPLS NCP protocol is enabled but is currently closed. MPLS NCP traffic is dropped.
pppStatusNegotiation	2	The MPLS NCP protocol is currently being negotiated on the link. This state may continue indefinitely if the peer refuses to negotiate MPLS NCP.
pppStatusOpen	3	The MPLS NCP protocol is currently open and IPCP traffic may flow.

osiState

Read-only. The current state of the OSI Network Control Protocol negotiation.

State	Value	Usage
pppStatusDisabled	0	The OSI NCP protocol has been disabled and is not negotiated during the NCP phase.
pppStatusClose	1	The OSI NCP protocol is enabled but is currently closed. OSI NCP traffic is dropped.
pppStatusNegotiation	2	The OSI NCP protocol is currently being negotiated on the link. This state may continue indefinitely if the peer refuses to negotiate OSI NCP.
pppStatusOpen	3	The OSI NCP protocol is currently open and IPCP traffic may flow.

peerIPAddress

Read-only. The negotiated IP address of the peer.

peerIpV6Iid

Read-only. The negotiated IPV6 Interface Id of the peer.

rxAlignment

Read-only. The negotiated OSI receive alignment.

rxMaxReceiveUnit

Read-only. Maximum frame size in receive direction. (*default = 0*)

rxMagicNumberStatus

Read-only. The status of receive magic number negotiation.

Value	Usage
a number	If a receive magic number has been negotiated, then its value is shown.
"Not Negotiated"	The receive magic number is not enabled in the ppp command.
"Disabled"	The peer does not agree to negotiate a receive magic number.
"Enabled"	The peer agrees to negotiate and the negotiation is in progress.

txAlignment

Read-only. The negotiated OSI receive alignment.

txMagicNumberStatus

Read-only. The status of transmit magic number negotiation.

Value	Usage
a number	If a transmit magic number has been negotiated, then its value is shown.
"Not Negotiated"	The transmit magic number is not enabled in the ppp command.
"Disabled"	The peer does not agree to negotiate a transmit magic number.
"Enabled"	The peer agrees to negotiate and the negotiation is in progress.

txMaxReceiveUnit

Read-only. Maximum frame size in transmit direction. (*default = 0*)

useMagicNumberRx

Read-only. The current state of the receive magic number negotiation.

State	Value	Usage
pppStatusDisabled	0	The negotiation of received Magic Number has been disabled and is not negotiated during the NCP phase.
pppStatusClose	1	The negotiation of received Magic Number is enabled but is currently closed. Related traffic is dropped.
pppStatusNegotiation	2	The received Magic Number is currently being negotiated on the link. This state may continue indefinitely if the peer refuses to negotiate.
pppStatusOpen	3	The negotiation of received Magic Number is currently open and related traffic may flow.

useMagicNumberTx

Read-only. The current state of the transmit magic number negotiation.

State	Value	Usage
pppStatusDisabled	0	The negotiation of transmitted Magic Number has been disabled and is not negotiated during the NCP phase.
pppStatusClose	1	The negotiation of transmitted Magic Number is enabled but is currently closed. Related traffic is dropped.
pppStatusNegotiation	2	The transmitted Magic Number is currently being negotiated on the link. This state may continue indefinitely if the peer refuses to negotiate.
pppStatusOpen	3	The negotiation of transmitted Magic Number is currently open and related traffic may flow.

COMMANDS

The **pppStatus** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

pppStatus cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **pppStatus** command.

pppStatus config option value

Modify the configuration options. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **pppStatus**.

pppStatus get chasID cardID portID [circuitID]

Gets the current PPP Status information on port with circuit *circuitID*, id *portID* on card *cardID*, chassis *chasID* from its hardware. Call this command before calling **pppStatus cget option value** to get the value of the configuration option. If *circuitID* = 0, gets information for the port; if *circuitID* not 0, gets information for the circuit.

EXAMPLES

See examples under [ppp](#)

SEE ALSO

[ppp](#)

NAME - prbsCapture

prbsCapture - captures PRBS packets on a port.

SYNOPSIS	prbsCapture <i>sub-command options</i>
-----------------	--

DESCRIPTION	The prbsCapture command is enabled on a per-port basis for capture of PRBS packets. Wide packet group must be enabled when using PRBS.
--------------------	---

STANDARD OPTIONS

referencePacket	Hex representation of the current frameNumber's good packet data (<i>default= 0</i>)
receivedPacket	Hex representation of the current frameNumber's bad packet data (<i>default= “”</i>)
numPackets	Total number of packets that are available in the PRBS capture buffer. (<i>default= “”</i>)
timestamp	Packet arrival time. (<i>default= 0</i>)

COMMANDS	The prbsCapture command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

prbsCapture **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **prbsCapture** command.

prbsCapture **config** *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS).

prbsCapture **setDefault**

Sets to IxTclHal default values for all configuration options.

prbsCapture **get** *chassisID cardID portID from to*

Retrieves the captured PRBS data from the server.

prbsCapture **getPacket** *packetNum*

Brings into scope a particular frame out of all the retrieved captured PRBS data.

EXAMPLES

```
class TCLPrbsCapture
{
public:
    TCLPrbsCapture();
    virtual ~TCLPrbsCapture();
```

```
VOID setDefault();
    INT get(INT chassisID, INT cardId, INT portId, INT from = 1,
INT to = 1 );
    INT getPacket( INT packetNum );

    // read-only parameters
    hexstring referencePacket;
    hexstring receivedPacket;
    int      numPackets;
    __int64   timestamp;

private:
    Copy(???);
};
```

SEE ALSO

port (receiveMode >portRxPrbs) (isValidFeature > portFeaturePRBS)
autoDetectInstrumentation (enablePrbs)
packetGroupStats (prbsBitsReceived, prbsErroredBits, and prbsBerRatio)
portGroup (clearPrbsCapture, startPrbsCapture, stopPrbsCapture)
stat (prbsFramesReceived, prbsHeaderError, prbsBitsReceived, prbsErroredBits, and prbsBerRatio)

NAME - protocol

protocol - configure the type of protocol to use for running the tests.

SYNOPSIS

protocol *sub-command options*

DESCRIPTION

This command allows to select the ethernet frame type and protocol type to use when building data packets or running a test.

Note: To set these values in IxHal and to commit them to the hardware use **stream set** and **stream write**.

STANDARD OPTIONS

appName

The application running on top of IP. These are applications may be layer 3 or 5 and others that cannot be directly set in the IP header. To use layer 4 applications such as UDP and TCP, use **ip config –ipProtocol** command. Available options are:

Option	Value	Usage
Udp	5	Tests UDP protocol
Arp	8	Tests ARP protocol
Rip	11	Tests RIP protocol
Dhcp	13	Tests DHCP protocol
SrpDiscovery	41	Tests SRP Discovery protocol
SrpArp	42	Tests SRP ARP protocol
SrpIps	43	Tests SRP IPS protocol
RprTopology	47	Tests RPR Topology protocol
RprProtection	48	Tests RPR Protection protocol
RprOam	49	Tests RPR OAM protocol
Ptp	72	Precision Time Protocol

enable802dot1qTag *true/false*

Sets the type of 802.1q Vlan tagged frame insertion.

Option	Value	Usage
vlanNone	0	(default) No VLANs used.
vlanSingle	1	A single VLAN specification is used.
vlanStacked	2	Two or more VLANs in a stack are used, as set in the stackedVlan command.

enableCMD *true/false*

Enable Cisco Metadata tagged frame insertion. (*default = false*)

enableDataCenterEnca psulation *true/false*

Enable Data Center Encapsulation option. (*default = false*)

enableISLtag *true/false*

Enable Cisco ISL tagged frame insertion. (*default = false*)

enableMacSec *true/false*

Enable MacSec frame insertion in streams. (*default = false*)

A**protocol**

enableMPLS *true/false* Enable MPLS Tagged frame insertion. (*default = false*)

enableOAM *true/false* Enable OAM frame insertion in streams. (*default = false*)

enableProtocolPad *true/false* If true, enables Protocol Pad.

etherType The type of ethernet frame selected. Options include:

Option	Value	Usage
noType	0	(<i>default</i>)
ethernetII	1	Ethernet II type of ethernet frame selected
ieee8023snap	2	IEEE8023 snap type of ethernet frame selected
ieee8023	3	IEEE8023 type of ethernet frame selected
ieee8022	15	IEEE8022 type of ethernet frame selected
protocolOffsetType	53	The protocol offset type of ethernet frame selected

name

The name of the protocol selected. Options include:

Option	Value	Usage
mac	0	MAC layer 2. During the learn process, simple MAC frames that contain the MAC address of the receive ports is transmitted to allow the switch to learn the ports (<i>default</i>)
ip	4	Uses an IP version 4 header in the frame, see ip command set. If name is set to ip, during the learn process ARP frames from both the transmit and receive ports is sent to DUT. From the ARP frames, the DUT learns the IP address of the attached Ixia ports and the Ixia ports learns the MAC address of the DUT port.
ipV4	4	same as <i>ip</i> above.
ipx	7	Uses an IPX header in the frame, see ipx command set. During the learn process, RIPx frames both the transmit and receive ports is sent to DUT so it may learn the network address of the attached ports and so that the transmit ports may learn the MAC address of the attached DUT port.
pauseControl	12	Pause control protocol. See pauseControl for details on setting up a pause control packet.
ipV6	31	Uses an IP version 6 header in the frame.
fcoe	68	Uses an FCoE header in the frame.
nativeFc	74	Uses an Fibre Channel header in the frame.

DEPRECATED STANDARD OPTIONS

dutStripTag *true/false*

COMMANDS

The **protocol** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

protocol cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **protocol** command.

Note: Call command **stream get chasID cardID portID streamID** before calling **protocol cget option value** to get the value of the configuration option.

protocol config option value

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

protocol setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Set to ethernet II and IPv4
protocol setDefault
protocol config -name          IPv4
protocol config -ethernetType ethernetII

# Protocol values are saved via the stream command
stream set $chas $card $port 1
```

```
ixWriteConfigToHardware $portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

stream, ip, ipx, protocolOffset

NAME - protocolOffset

protocolOffset - configure the offset used to generate protocol header and contents.

SYNOPSIS	protocolOffset <i>sub-command options</i>
DESCRIPTION	<p>For load modules which support this feature, this allows the protocol headers and contents to be generated at other than the standard location (byte 14) within a packet.</p> <p>Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in atmHeader. The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command is with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.</p>
<hr/>	
STANDARD OPTIONS	
offset	The new offset for protocol header location. (<i>default = 14</i>)
userDefinedTag	The new contents for the old protocol header location (byte 14). If the tag is smaller than the space between the old and new offset, then zeroes are used to fill in the remainder. If the tag is larger than the space, it is truncated. (<i>default = {00 00}</i>)
<hr/>	
COMMANDS	<p>The protocolOffset command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.</p> <p>protocolOffset cget <i>option</i> Returns the current value of the configuration option given by <i>option</i>. <i>Option</i> may have any of the values accepted by the protocolOffset command.</p> <p>protocolOffset config <i>option value</i> Modify the configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.</p> <p>protocolOffset set <i>chasID cardID portID</i> Sets the configuration of the protocol offset in IxHAL on port with id <i>portID</i> on card <i>cardID</i>, chassis <i>chasID</i> by reading the configuration option values set by the protocolOffset config <i>option value</i> command. Specific errors are:</p> <ul style="list-style-type: none"> • No connection to a chassis • Invalid port number • The port is being used by another user • The configured parameters are not valid for this port

protocolOffset setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
protocolOffset setDefault
protocol config -ethernetType protocolOffsetType
protocolOffset config -offset          20
protocolOffset config -userDefinedTag {01 02 03 04 05 06}

if [protocolOffset set $chas $card $port] {
    ixPuts "Error in protocolOffset set for $chas $card $port"
}
```

SEE ALSO

ip, ipx, protocol

NAME - protocolServer

protocolServer - use to enable the various protocols.

SYNOPSIS	protocolServer <i>sub-command options</i>
DESCRIPTION	For load modules which support this feature, this enables the protocols listed in Standard Options, below.
STANDARD OPTIONS	
enableArpResponse <i>true/false</i>	ARP must be enabled in protocolServer in order for ARP to work. (<i>default = false</i>)
enablePingResponse <i>true/false</i>	Ping must be enabled in protocolServer in order for Ping to work. (<i>default = false</i>)
enableIgmpQuery Response <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enableOspfService <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enableBgp4Service <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enableIsisService <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enableRsvpService <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enableRipService <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enableLdpService <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enableRipngService <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enableMldService <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enableOspfv3Service <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enablePimsmService <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)
enableBgp4Create Interface <i>true/false</i>	Configurable when IxTclProtocol is installed. (<i>default = false</i>)

enableIsisCreate
Interface true/false Configurable when IxTclProtocol is installed. (default = false)

enableOspfCreate
Interface true/false Configurable when IxTclProtocol is installed. (default = false)

enableRipCreate
Interface true/false Configurable when IxTclProtocol is installed. (default = false)

enableRsvpCreate
Interface true/false Configurable when IxTclProtocol is installed. (default = false)

enableIgmpCreate
Interface true/false Configurable when IxTclProtocol is installed. (default = false)

COMMANDS

The **protocolServer** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

protocolServer **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **protocolServer** command.

protocolServer **config** *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

protocolServer **get** *option*

Retrieves the current configuration of the protocol server for *option*.

protocolServer **set** *chasID cardID portID*

Sets the configuration of the protocol server in IxHAL on port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **protocolServer config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

protocolServer **setDefault** *option*

Sets to IxTclHal default values for all configuration options.

protocolServer **write** *chasID cardID portID*

Writes or commits the changes in IxHAL to hardware the protocol server configuration for each port with id *portID* on card *cardID*, chassis *chasID*.

Before using this command, use the **protocolServer set** command to configure the port related parameters in IxHAL. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

**DEPRECATED
OPTIONS**

arpServerEnable
true/false

pingServerEnable
true/false

repeatCount
true/false

rate (in pps)

MacAddress

IpAddress

count

mapType

EXAMPLES

See [arp](#).

SEE ALSO

NAME - protocolPad

protocolPad - configures data bytes.

SYNOPSIS *protocolPad sub-command options*

DESCRIPTION This command, when true, allows to configure data bytes.

STANDARD OPTIONS

dataBytes When protocolPad option is enabled in protocol object, it allows to configure data bytes using the "config dataBytes" command. The value maybe as follows:

11 22 33 44 55 66

COMMANDS The **protocolPad** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

protocol set option

Sets the protocolPad option.

protocol get option

Gets the protocolPad option.

NAME - ptp

ptp- configure Precision Time Protocol to synchronize clocks.

SYNOPSIS

`ptp sub-command options`

DESCRIPTION

Precision Time Protocol (PTP) enables precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing, and distributed objects. The protocol is applicable to systems communicating by local area networks supporting multicast messaging including, but not limited to, Ethernet. The protocol enables heterogeneous systems that include clocks of various inherent precision, resolution, and stability to synchronize to a grandmaster clock. The protocol supports system-wide synchronization accuracy in the sub-microsecond range with minimal network and local clock computing resources.

The Sync, Delay_Req, Follow_Up, and Delay_Resp messages are used to generate and communicate the timing information needed to synchronize ordinary and boundary clocks using the delay request-response mechanism.

STANDARD OPTIONS

controlField

The value of controlField depends on the message type defined in the *messageType* field. (*default = 5*)

Option	Value	Usage
ptpSync	0	Sync
ptpDelayRequest	1	Delay request
ptpFollowUp	2	Follow-up
ptpDelayResponse	3	Delay response
ptpManagement	4	Management
ptpOther	5	(<i>default</i>) Other

correctionField

Transparent clocks forward PTP timing messages through the clock in the manner of an ordinary bridge or router but, in addition, measure the time spent by a PTP timing message within the transparent clock. These “residence” times are accumulated in the *correctionField* in the PTP timing messages, which allows the slave to correct the timestamps, effectively removing the timing fluctuations that would otherwise be introduced by the bridges. Expressed in nanoseconds and fractions thereof. (*default = 0.0*)

domainNumber

A domain consists of one or more PTP devices communicating with each other as defined by the protocol. Range 0 - 255. (*default = 0*)

Value	Usage
0	(<i>default</i>)
1	Alternate domain 1
2	Alternate domain 2

Value	Usage
3	Alternate domain 2
4-127	User-defined
128-255	Reserved

extensionId

Extension identifier. 5 byte hex list. (*default = "00 00 00 00 00"*)

flagField

ORed values from *flagField* array, below. (*default = 0*)

Example code:

```
[expr
$::ptpUtcOffsetValid|$::ptpFrequencyTraceable|$::ptpTwoStep|$::ptpUnicast
```

Option	Value	Usage
ptpLeap61	1	Leap 61
ptpLeap59	2	Leap 59
ptpUtcOffsetValid	4	UTC offset valid
ptpPtpTimescale	8	PTP timescale
ptpTimeTraceable	16	Time traceable
ptpFrequencyTraceable	32	Frequency traceable
ptpAlternateMaster	256	Alternate master
ptpTwoStep	512	Two step For a one-step clock, the value of twoStepFlag shall be FALSE. For a two-step clock, the value of twoStepFlag shall be TRUE.
ptpUnicast	1024	Unicast TRUE, if the transport layer protocol address to which this message was sent is a unicast address. FALSE, if the transport layer protocol address to which this message was sent is a multicast address.
ptpProfile1	8192	Profile 1
ptpProfile2	16384	Profile 2

logMessageInterval

The value of the *logMessageInterval* field is determined by the type of the message. (*default = 0*)

messageLength

Read only. The total number of octets that form the PTP message. The counted octets start with the first octet of the header and include and terminate with the last octet of any suffix or, if there are no suffix members with the last octet of the message. (*default = 44*)

messageType

Configure the message type from list. (*default = ptpSyncMessage*)

Option	Value	Usage
ptpSyncMessage	0	PTP sync message
ptpDelayRequestMessage	1	PTP delay request message
ptpFollowUpMessage	8	PTP follow-up message
ptpDelayResponseMessage	9	PTP delay response message
ptpAnnounceMessage	11	PTP announce message

organizationUniqueId Organization Unique Identifier (OUI): the value of the OUI assigned to the vendor or standards organization by the IEEE. The most significant 3 octets of the clockIdentity shall be an OUI. (*default = "00 00 00"*)

portNumber Identifies a specific Precision Time Protocol (PTP) port on a PTP node.

sequenceId The *sequenceId* of the message shall be one greater than the *sequenceId* of the previous message of the same message type sent to the same message destination address by the transmitting port. (*default = 0*)

transportSpecific *Read only*. The transportSpecific field (*default = 0*)

Bit	Name	Meaning
0	hardwareCompatibility	Check the length of the incoming packet before qualifying the timestamp and require the UDP payload of the PTP event messages to be at least 124 octets in length. Nodes using such hardware shall set bit 0 equal to "1" in all Announce and PTP event messages transmitted from the node.
1-3	reserved	The bit shall be transmitted as zero and ignored by the receiver

version *Read only*. Displays the PTP version. (*default = 2*)

COMMANDS

The **ptp** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ptp cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ptp** command.

ptp config option value

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **ptp**.

ptp decode capFrame chasID cardID portID

Decodes the **ptp** header and trailer packet and refreshes the IxTclHal object.

ptp get chasID cardID portID

Gets current **ptp** header and trailer settings from IxHal and refreshes IxTclHal object.

ptp set chasID cardID portID

Sets the current **ptp** header and trailer settings from IxTclHal to local IxHal. Specific errors are:

- No connection to a chassis
- Invalid port number
- Unsupported feature
- The port is being used by another user
- The configured parameters are not valid for this port

ptp setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package req IxTclHal

set hostname loopback

if {[ixConnectToChassis $hostname]} {
    errorMsg "error connecting $hostname chassis"
    return "FAIL"
}
set chassisId [chassis cget -id]
set cardId 20
set portId 1
set streamId 1

set portList [list [list $chassisId $cardId $portId] ]

if { [port isValidFeature $chassisId $cardId $portId $::portFeaturePtp] } {

    # Configure Ptp streams
    protocol setDefault
    protocol config -ethernetType $::ethernetII
    protocol config -name $::ipV4
    protocol config -appName $::Ptp
    ip setDefault
    ip config -sourceIpAddr "111.111.112.222"
    if {[ip set $chassisId $cardId $portId]} {
        errorMsg "Error calling ip set $chassisId $cardId $portId"
        return "FAIL"
    }
    udp setDefault
    udp config -sourcePort ptpEventPort
    udp config -destPort ptpGeneralPort
    if {[udp set $chassisId $cardId $portId]} {
        errorMsg "Error calling udp set $chassisId $cardId $portId"
        return "FAIL"
    }
    ptpAnnounce setDefault
    ptpAnnounce config -seconds 1111
    ptpAnnounce config -nanoseconds 9999999
    ptpAnnounce config -currentUtcOffset 1236
    ptpAnnounce config -stepsRemoved 55
    ptpAnnounce config -timeSource $::ptpAltPtpProfile0
    ptpAnnounce config -priority1 11
    ptpAnnounce config -priority2 12
    ptpAnnounce config -clockClass 99
    ptpAnnounce config -clockAccuracy $::ptpAccuracy100us
    ptpAnnounce config -clockLogVariance 0
    ptpAnnounce config -extensionId "AB CD EF 12 34"
    ptpAnnounce config -organizationUniqueId "10 11 12"
    ptp setDefault
    ptp config -controlField $::ptpDelayRequest
    ptp config -logMessageInterval 33
    ptp config -domainNumber 55
    ptp config -correctionField 123654.0
    ptp config -sequenceId 6
```

```

ptp config -flagField [expr
$:::ptpUtcOffsetValid|$:::ptpFrequencyTraceable|$:::ptpTwoStep|$:::ptpUnicast]
    ptp config -messageType $:::ptpAnnounceMessage
    ptp config -portNumber 9999
    ptp config -extensionId "AA D4 5D FE ED"
    ptp config -organizationUniqueId "12 AA 45"
    if {[ptp set $chassisId $cardId $portId]} {
        errorMsg "Error setting ptp on $chassisId $cardId $portId"
        return "FAIL"
    }

    streamsetDefault
    stream config -name "Ptp Announce Stream"
    if {[stream set $chassisId $cardId $portId $streamId]} {
        errorMsg "Error setting stream on $chassisId $cardId $portId $streamId"
        return "FAIL"
    }

    incr streamId

    ptpDelayResponsesetDefault
    ptpDelayResponse config -seconds 999
    ptpDelayResponse config -nanoseconds 11
    ptpDelayResponse config -portNumber 555
    ptpDelayResponse config -extensionId "00 AB CD 12 34"
    ptpDelayResponse config -organizationUniqueId "55 EF DA"
    ptp setDefault
    ptp config -controlField $:::ptpDelayResponse
    ptp config -logMessageInterval 127
    ptp config -domainNumber 255
    ptp config -correctionField 8888888
    ptp config -sequenceId 2
    ptp config -flagField [expr $:::ptpTwoStep | $:::ptpUtcOffsetValid |
$:::ptpFrequencyTraceable]
    ptp config -messageType $:::ptpDelayResponseMessage
    ptp config -portNumber 0
    ptp config -extensionId "00 34 AB 33 33"
    ptp config -organizationUniqueId "B2 22 2A"
    if {[ptp set $chassisId $cardId $portId]} {
        errorMsg "Error setting ptp on $chassisId $cardId $portId"
        return "FAIL"
    }
    streamsetDefault
    stream config -name "Ptp DelayResponse Stream"
    if {[stream set $chassisId $cardId $portId $streamId]} {
        errorMsg "Error setting stream on $chassisId $cardId $portId $streamId"
        return "FAIL"
    }

# Configure PTP interfaces
if {[interfaceTable select $chassisId $cardId $portId]} {
    errorMsg "Error selecting interfaceTable on $chassisId $cardId $portId.."
    return "FAIL"
}
ptpPropertiessetDefault
ptpProperties config -clockId "AA 00 00 00 00 00 00 BC"
ptpProperties config -portNumber 22
ptpProperties config -enableClockMaster $:::true
ptpProperties config -timestampError 11
ptpProperties config -badCrcPercent 0
ptpProperties config -dropFollowUpPercent 11
ptpProperties config -dropDelayResponsePercent 99
interfaceEntry config -enable false

```

```

interfaceEntry config -description {ProtocolInterface - 27:01 - 1}
interfaceEntry config -enablePtp true
if {[interfaceTable addInterface interfaceTypeConnected]} {
    errorMsg "Error adding interfaceTypeConnected to interfaceTable on $chassisId $cardId
$portId."
    return "FAIL"
}
set interfaceDescription [interfaceEntry cget -description ]
ixWriteConfigToHardware portList

# Example how to retrieve PTP discovered information
if {[interfaceTable select $chassisId $cardId $portId]} {
    errorMsg "Error selecting interfaceTable on $chassisId $cardId $portId."
    return "FAIL"
}
interfaceEntry setDefault
ptpProperties setDefault
if {[interfaceTable getFirstInterface interfaceTypeConnected]} {
    errorMsg "Error adding interfaceTypeConnected to interfaceTable on $chassisId $cardId
$portId."
    return "FAIL"
}
ixPuts "enablePtp: [interfaceEntry cget -enablePtp]"
ixPuts "announceInterval: [ptpProperties cget -announceInterval]"

# Below code is just for usage example
#if {[interfaceTable requestDiscoveredTable]} {
#    errorMsg "Error interfaceTable requestDiscoveredTable on $chassisId $cardId $portId."
#    return "FAIL"
#
# Some delay before the discovered information is ready, may depend on the configuration
#after 2000
#ptpDiscoveredInfo setDefault
#if {[interfaceTable getPtpDiscoveredInfo $interfaceDescription]} {
#    errorMsg "Error getting PTP Discovered table for $interfaceDescription on $chassisId
$cardId $portId."
#    return "FAIL"
#
#ixPuts "ptpDiscoveredInfo clockId [ptpDiscoveredInfo cget -clockId]"
#ixPuts "ptpDiscoveredInfo announceMessageSent [ptpDiscoveredInfo cget -
announceMessageSent]"
#ixPuts "ptpDiscoveredInfo timeStamp [ptpDiscoveredInfo cget -timeStamp]"

} else {

    errorMsg "portFeaturePtp is not valid on $chassisId $cardId $portId"
    return "FAIL"
}

```

SEE ALSO

*ptpAnnounce, ptpDelayRequest, ptpProperties, ptpFollowUp,
ptpDelayResponse, ptpSync, ptpDiscoveredInfo*

NAME - ptpAnnounce

ptpAnnounce - configure PTP Announce message.

SYNOPSIS

ptpAnnounce *sub-command options*

DESCRIPTION

Announce messages are periodically sent by one port and delivered to all other ports of ordinary or boundary clocks within a communication path. The Announce message is used to establish the synchronization hierarchy. Announce messages provide status and characterization information of the transmitting node and its grandmaster. This information is used by the receiving node when executing the best master clock algorithm.

If the port is in the master state and the ordinary clock is the grandmaster clock of the domain, then the local clock is typically synchronized to an external source of time traceable to International Atomic Time (TAI) and UTC (Coordinated Universal Time) such as the GPS system.

STANDARD OPTIONS

clockAccuracy

Defines the accuracy of a clock. (*default = ptpAccuracyUnknown*)

Option	Value	Usage
ptpAccuracy25ns	32	accuracy 25 nanoseconds
ptpAccuracy100ns	33	accuracy 100 ns
ptpAccuracy250ns	34	accuracy 250 ns
ptpAccuracy1us	35	accuracy 1 microsecond
ptpAccuracy2p5us	36	accuracy 2.5 microseconds
ptpAccuracy10us	37	accuracy 10 microseconds
ptpAccuracy25us	38	accuracy 25 microseconds
ptpAccuracy100us	39	accuracy 100 microseconds
ptpAccuracy250us	40	accuracy 250 microseconds
ptpAccuracy1ms	41	accuracy 1 millisecond
ptpAccuracy2p5ms	42	accuracy 2.5 milliseconds
ptpAccuracy10ms	43	accuracy 10 milliseconds
ptpAccuracy25ms	44	accuracy 25 milliseconds
ptpAccuracy100ms	45	accuracy 100 milliseconds
ptpAccuracy250ms	46	accuracy 250 milliseconds
ptpAccuracy1s	47	accuracy 1 second
ptpAccuracy10s	48	accuracy 10 seconds
ptpAccuracyGreater10s	49	accuracy greater than 10 seconds
ptpAccuracyUnknown	254	(default) accuracy unknown

clockClass

Defines a clock's TAI traceability. The *clockClass* attribute of an ordinary or boundary clock denotes the traceability of the time or frequency distributed by the grandmaster clock. (*default = 0*)

clockLogVariance

Defines the stability of a clock. (*default = 0*)

currentUtcOffset	Current UTC offset. The UTC time differs from the TAI time by a constant offset. This is calculated as follows: TAI - UTC. (<i>default = 0</i>)
extensionId	Extension identifier. 5 byte hex list. (<i>default = "00 00 00 00 00"</i>)
nanoSeconds	The time interval, expressed in nanoseconds. (<i>default = 0</i>)
organizationUniqueId	Organization Unique Identifier (OUI): the value of the OUI assigned to the vendor or standards organization by the IEEE. The most significant 3 octets of the clockIdentity shall be an OUI.. (<i>default = "00 00 00"</i>)
priority1	A user configurable designation that a clock belongs to an ordered set of clocks from which a master is selected. (<i>default = 0</i>)
priority2	A user configurable designation that provides finer grained ordering among otherwise equivalent clocks. (<i>default = 0</i>)
stepsRemoved	In addition to this precedence order, the distance measured by the number of boundary clocks between the local clock and the foreign master is used when two Announce messages reflect the same foreign master. (<i>default = 0</i>)
timeSource	Indicates the source of time used by the grandmaster clock. <i>(default = ptpTimeSourceOther)</i>

Option	Value	Usage
ptpAtomicClock	16	atomic clock
ptpGPS	32	GPS
ptpTerrestrialRadio	48	terrestrial radio
ptpPTP	64	PTP
ptpNTP	80	NTP
ptpHandSet	96	handset
ptpTimeSourceOther	144	(default) time source other
ptpInternalOscillator	160	internal oscillator
ptpAltPtpProfile0	240	alt ptp profile 0
ptpAltPtpProfile1	241	alt ptp profile 1
ptpAltPtpProfile2	242	alt ptp profile 2
ptpAltPtpProfile3	243	alt ptp profile 3
ptpAltPtpProfile4	244	alt ptp profile 4
ptpAltPtpProfile5	245	alt ptp profile 5
ptpAltPtpProfile6	246	alt ptp profile 6
ptpAltPtpProfile7	247	alt ptp profile 7
ptpAltPtpProfile8	248	alt ptp profile 8
ptpAltPtpProfile9	249	alt ptp profile 9
ptpAltPtpProfile10	250	alt ptp profile 10
ptpAltPtpProfile11	251	alt ptp profile 11
ptpAltPtpProfile12	252	alt ptp profile 12
ptpAltPtpProfile13	253	alt ptp profile 13
ptpAltPtpProfile14	254	alt ptp profile 14
ptpReserved	255	reserved

COMMANDS

The **ptpAnnounce** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

ptpAnnounce cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ptpAnnounce** command.

ptpAnnounce config *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **ptpAnnounce**.

ptpAnnounce setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under [ptp](#) command.

SEE ALSO

[ptp](#), [ptpDelayRequest](#), [ptpProperties](#), [ptpFollowUp](#), [ptpDelayResponse](#), [ptpSync](#), [ptpDiscoveredInfo](#)

NAME - ptpDelayRequest

ptpDelayRequest - configure PTP Delay Request messages.

SYNOPSIS	ptpDelayRequest <i>sub-command options</i>
DESCRIPTION	The Sync, Delay_Req, Follow_Up, and Delay_Resp messages are used to generate and communicate the timing information needed to synchronize ordinary and boundary clocks using the delay request-response mechanism.
<hr/>	
STANDARD OPTIONS	
nanoseconds	The time interval, expressed in nanoseconds. (<i>default = 0</i>)
seconds	The time interval, expressed in seconds. (<i>default = 0</i>)
<hr/>	
COMMANDS	The ptpDelayRequest command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
ptpDelayRequest cget <i>option</i>	
Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the ptpDelayRequest command.	
ptpDelayRequest config <i>option value</i>	
Modify the configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for ptpDelayRequest .	
ptpDelayRequest setDefault	
Sets to IxTclHal default values for all configuration options.	
<hr/>	
EXAMPLES	See example under ptp command.
<hr/>	
SEE ALSO	ptp , ptpAnnounce , ptpProperties , ptpFollowUp , ptpDelayResponse , ptpSync , ptpDiscoveredInfo

NAME - ptpDelayResponse

ptpDelayResponse - configure PTP Delay Response messages.

SYNOPSIS	ptpDelayResponse <i>sub-command options</i>
-----------------	---

DESCRIPTION	The Sync, Delay_Req, Follow_Up, and Delay_Resp messages are used to generate and communicate the timing information needed to synchronize ordinary and boundary clocks using the delay request-response mechanism.
--------------------	--

STANDARD OPTIONS

extensionId	Extension identifier. 5 byte hex list. (<i>default</i> = “00 00 00 00 00”)
nanoseconds	The time interval, expressed in nanoseconds. (<i>default</i> = 0)
organizationUniqueId	Organization Unique Identifier (OUI): the value of the OUI assigned to the vendor or standards organization by the IEEE. The most significant 3 octets of the clockIdentity shall be an OUI. (<i>default</i> = “00 00 00”)
portNumber	16-bit port number associated with the clock. (<i>default</i> = 0)
seconds	The time interval, expressed in seconds. (<i>default</i> = 0)

COMMANDS	The ptpDelayResponse command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	---

ptpDelayResponse **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ptpDelayResponse** command.

ptpDelayResponse **config** *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **ptpDelayResponse**.

ptpDelayResponse **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES	See example under ptp command.
-----------------	--

SEE ALSO	ptp , ptpAnnounce , ptpDelayRequest , ptpProperties , ptpFollowUp , ptpSync , ptpDiscoveredInfo
-----------------	---

NAME - ptpDiscoveredInfo

ptpDiscoveredInfo - configure PTP discovery function.

SYNOPSIS	ptpDiscoveredInfo <i>sub-command options</i>
DESCRIPTION	PTP ports discover other ports within a communication path through the receipt of multicast Announce messages.
STANDARD OPTIONS	
announceMessage Received	<i>Read only</i> . Number of announce messages received by the interface. (<i>default = 0</i>)
announceMessageSent	<i>Read only</i> . Number of announce messages sent by the interface. (<i>default = 0</i>)
clockId	<i>Read only</i> . Identifies a clock. (<i>default = '00 00 00 00 00 00 00 00'</i>)
clockOffset	<i>Read only</i> . The offset of the slave clock in nanoseconds with reference to its master, as calculated by the slave per 1588 protocol. It is a measure of time transfer. (<i>default = 0</i>)
delayRequestMessage Received	<i>Read only</i> . Number of delay request messages received by the interface. (<i>default = 0</i>)
delayRequestMessage Sent	<i>Read only</i> . Number of delay request messages sent by the interface. (<i>default = 0</i>)
delayResponseMessage Received	<i>Read only</i> . Number of delay response messages received by the interface. (<i>default = 0</i>)
delayResponseMessage Sent	<i>Read only</i> . Number of delay response messages sent by the interface. (<i>default = 0</i>)
followupMessage Received	<i>Read only</i> . Number of follow-up messages received by the interface. (<i>default = 0</i>)
followupMessageSent	<i>Read only</i> . Number of follow-up messages sent by the interface. (<i>default = 0</i>)
meanPathDelay	<i>Read only</i> . The mean propagation time between master and slave clock as computed by the slave. (<i>default = 0</i>)
syncMessageReceived	<i>Read only</i> . Number of sync messages received by the interface. (<i>default = 0</i>)
syncMessageSent	<i>Read only</i> . Number of sync messages sent by the interface. (<i>default = 0</i>)
timeSlope	<i>Read only</i> . The ratio of the slave clock frequency to its master clock frequency. It is a measure of frequency transfer. (<i>default = 0</i>)
timeStamp	<i>Read only</i> . Timestamp of statistics. (<i>default = 0</i>)

COMMANDS

The **ptpDiscoveredInfo** command is invoked with the following sub-commands.
If no sub-command is specified, returns a list of all sub-commands available.

ptpDiscoveredInfo *cget option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ptpDiscoveredInfo** command.

ptpDiscoveredInfo *config option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **ptpDiscoveredInfo**.

ptpDiscoveredInfo *setDefault*

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example under *ptp* command.

SEE ALSO

ptp, ptpAnnounce, ptpDelayRequest, ptpDelayResponse, ptpProperties, ptpFollowUp, ptpSync,

NAME - ptpFollowUp

ptpFollowUp - configure PTP FollowUp messages.

SYNOPSIS *ptpFollowUp sub-command options*

DESCRIPTION The Sync, Delay_Req, Follow_Up, and Delay_Resp messages are used to generate and communicate the timing information needed to synchronize ordinary and boundary clocks using the delay request-response mechanism.

STANDARD OPTIONS

nanoseconds The time interval, expressed in nanoseconds. (*default = 0*)

seconds The time interval, expressed in seconds. (*default = 0*)

COMMANDS The **ptpFollowUp** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.**ptpFollowUp cget *option***

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ptpFollowUp** command.

ptpFollowUp config *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **ptpFollowUp**.

ptpFollowUp setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES See example under *ptp* command.

SEE ALSO *ptp, ptpAnnounce, ptpDelayRequest, ptpDelayResponse, ptpDiscoveredInfo, ptpProperties*

NAME - ptpProperties

ptpProperties - configure .

SYNOPSIS

ptpProperties *sub-command options*

DESCRIPTION

There are five types of PTP devices, as follows:

- Ordinary clock
- Boundary clock
- End-to-end transparent clock
- Peer-to-peer transparent clock
- Management node

All PTP devices are identified by a *clockIdentity* attribute. In addition, ordinary and boundary clocks are characterized by the following attributes:

- priority1
- priority2
- clockClass
- clockAccuracy (see Standard Options below and *ptpAnnounce*)
- timeSource (see *ptpAnnounce*)
- clockLogVariance (see *ptpAnnounce*)
- portNumber

STANDARD OPTIONS

announceInterval

The mean time interval between successive Announce messages. (*default* = 0)

announceReceipt

The number of *announceInterval* that has to pass without receipt of an Announce message before the occurrence of the Announce Receipt Timeout event. (*default* = 0)

badCrcPercent

Percent of follow-up messages sent with bad CRC. (*default* = 0)

badTimeStampPercent

Percent of follow-up messages sent with bad timestamp. Bad timestamp = good timestamp + timestamp error. (*default* = 0)

clockAccuracy

Defines the accuracy of a clock. (*default* = *ptpAccuracy25ns*)

Option	Value	Usage
ptpAccuracy25ns	32	accuracy 25 nanoseconds
ptpAccuracy100ns	33	accuracy 100 ns
ptpAccuracy250ns	34	accuracy 250 ns
ptpAccuracy1us	35	accuracy 1 microsecond
ptpAccuracy2p5us	36	accuracy 2.5 microseconds
ptpAccuracy10us	37	accuracy 10 microseconds
ptpAccuracy25us	38	accuracy 25 microseconds
ptpAccuracy100us	39	accuracy 100 microseconds
ptpAccuracy250us	40	accuracy 250 microseconds
ptpAccuracy1ms	41	accuracy 1 millisecond

Option	Value	Usage
ptpAccuracy2p5ms	42	accuracy 2.5 milliseconds
ptpAccuracy10ms	43	accuracy 10 milliseconds
ptpAccuracy25ms	44	accuracy 25 milliseconds
ptpAccuracy100ms	45	accuracy 100 milliseconds
ptpAccuracy250ms	46	accuracy 250 milliseconds
ptpAccuracy1s	47	accuracy 1 second
ptpAccuracy10s	48	accuracy 10 seconds
ptpAccuracyGreater10s	49	accuracy greater than 10 seconds
ptpAccuracyUnknown	254	accuracy unknown

clockClass

Defines a clock's TAI traceability. The `clockClass` attribute of an ordinary or boundary clock denotes the traceability of the time or frequency distributed by the grandmaster clock. (*default = 0*)

clockId

Clock identity, identifies a clock. (*default = '00 00 00 00 00 00 00 00'*)

delayMechanism

Configure the the delay mechanism. (*default = ptpE2E*)

Option	Value	Usage
ptpE2E	1	End-to-end
ptpDisabled	254	Disabled

delayRequest

The minimum permitted mean time interval between successive Delay_Req messages, sent by a slave to a specific port on the master. (*default = 0*)

domainNumber

The domain is identified by an integer in the range of 0 to 255. (*default = 0*)

dropDelayResponsePercent

Defines how many delay response messages to be dropped. Drop delay response messages expressed as percentage of received delay request messges. Normally, delay response is sent by the master corresponding to each delay request massage received. For negative testing, you can configure Ixia port to drop the delay response message to see how the DUT behaves. (*default = 0*)

dropFollowUpPercent

Defines how many follow-up messages to be dropped. Drop follow-up messages expressed as percent of sync messages. Normally, a follow-up message is sent out corresponding to each sync message. For negative testing, you can configure Ixia port to drop the follow-up message to see how the DUT behaves. (*default = 0*)

enableClockMaster

If true, configures Ixia port in master mode. (*default = 0*)

portNumber

An index identifying a specific PTP port on a PTP node. (*default = 0*)

priority1

A user configurable designation that a clock belongs to an ordered set of clocks from which a master is selected. (*default = 0*)

priority2

A user configurable designation that provides finer grained ordering among otherwise equivalent clocks. (*default = 0*)

startOffset	Defines the clock offset in nanoseconds. Master sends PTP messages with Start Offset added to the clock. (<i>default</i> = 0)
syncInterval	The mean time interval between successive Sync messages. (<i>default</i> = 0)
timestampError	The time error between a slave and a master ordinary or boundary clock. (<i>default</i> = 0)

COMMANDS	The ptpProperties command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

ptpProperties cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ptpProperties** command.

ptpProperties config *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

ptpProperties setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES	See example under ptp command.
-----------------	--

SEE ALSO	ptp , ptpAnnounce , ptpDelayRequest , ptpDelayResponse , ptpDiscoveredInfo , ptpFollowUp , ptpSync
-----------------	--

NAME - ptpSync

ptpSync - configure PTP sync messages.

SYNOPSIS *ptpSync sub-command options*

DESCRIPTION The Sync, Delay_Req, Follow_Up, and Delay_Resp messages are used to generate and communicate the timing information needed to synchronize ordinary and boundary clocks using the delay request-response mechanism.

STANDARD OPTIONS

nanoseconds The time interval, expressed in nanoseconds. (*default = 0*)

seconds The time interval, expressed in seconds. (*default = 0*)

COMMANDS The **ptpSync** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.**ptpSync cget option**

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **ptpSync** command.

ptpSync config option value

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **ptpSync**.

ptpSync setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES See example under *ptp* command.

SEE ALSO *ptp, ptpAnnounce, ptpDelayRequest, ptpDelayResponse, ptpDiscoveredInfo, ptpFollowUp, ptpProperties*

NAME - qos

qos - configure the QoS counter parameters for a port

SYNOPSIS

qos *sub-command options*

DESCRIPTION

This command allows to set up the QoS counter filters and offset of the QoS priority bits.

Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in [atmHeader](#). The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command are with respect to the start of the AAL5 packet and must be adjusted by hand to account for the header.

STANDARD OPTIONS

byteOffset

The offset where the priority value is checked to indicate which of the QoS counters is going to be incremented. (*default = 14*)

packetType

The type of packet that the QoS counters are looking for priority bits within.

Option	Value	Usage
<i>ipEthernetII</i>	0	
<i>ip8023Snap</i>	1	
<i>vlan</i>	2	
<i>custom</i>	3	
<i>ipPPP</i>	4	
<i>ipCiscoHdlc</i>	5	
<i>ipAtm</i>	6	

patternMask

The mask of the pattern that is analyzed by the Receive engine to increment the QoS counter. (*default = 00 00*)

patternMatch

The pattern that is analyzed by the Receive engine to increment the QoS counter. (*default = 81 00*)

patternOffset

The offset where the pattern to be matched is located. (*default = 12*)

patternOffsetType

The point within a frame that *patternOffset* is with respect to.

Option	Value	Usage
<i>qosOffsetStartOfFrame</i>	0	(<i>default</i>) From the start of the frame.
<i>qosOffsetStartOfIp</i>	1	From the start of the IP header.
<i>qosOffsetStartOfProtocol</i>	2	From the start of the inner protocol header. For example, TCP header.
<i>qosOffsetStartOfSonet</i>	3	From the stat of the SONET frame.

COMMANDS

The **qos** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

qos cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **qos** command

qos config option value

Modify the configuration options of the qos. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for qos.

qos get chasID cardID portID

Gets the current configuration of the QoS counters on port with id *portID* on card *cardID*, chassis *chasID*. from its hardware. Call this command before calling **qos cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

qos set chasID cardID portID

Sets the configuration of the QoS counters in IxHAL on port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **qos config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

qos setDefault

Sets to IxTclHal default values for all configuration options.

qos setup packetType

Sets the QoS counters to look for priority bits for a certain type of packet. See the *packetType* standard option description for the choices. Specific errors are:

- Invalid packetType

qos write chasID cardID portID

Writes or commits the changes in IxHAL to hardware the QoS counters configuration for each port with id *portID* on card *cardID*, chassis *chasID*. Before using this command, use the **qos set** command to configure the port related parameters (byteOffset, patternMatch, patternMask, patternOffset) in IxHAL. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

EXAMPLES

```
package require IxTclHal
```

```
# In this test, we'll generate a number of packets with different
# settings in the QoS field. The directly connected receiving port
# will be set to receive and provide statistics for the number of
```

```

# QoS packets received at each of 8 levels

# Connect to chassis and get chassis ID
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assume card to be used is in slot 1
set card 1
set txPort 1
set rxPort 2
set portList [list [list $chas $card $txPort] \
                    [list $chas $card $rxPort] ]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Setup port 1 to transmit
port setFactoryDefaults $chas $card $txPort
port setDefault

# Stream: 100,000 packets
stream setDefault
stream config -numFrames      100000
stream config -dma           stopStream

# IP/ethernetII packets
ip setDefault
ip set $chas $card $txPort

protocol setDefault
protocol config -name          ipV4
protocol config -ethernetType   ethernetII

# Overlay UDF1 on top of the QoS byte with an appropriate mask
udf setDefault
udf config -enable              true
udf config -offset               15
udf config -udfSize              c8
udf config -continuousCount     true

```

```
udf config -maskselect          {1F 00 00 00}
udf set 1

stream set $chas $card $txPort 1
port set $chas $card $txPort

# Set up port 2 for QoS Statistics
port setFactoryDefaults $chas $card $rxPort
port setDefault

# QoS statistics mode
stat config -mode             statQos
stat set $chas $card $rxPort

# Set up locations of where to find the information
qos setup ipEthernetII
qos set $chas $card $rxPort

protocol setDefault
protocol config -name          mac
protocol config -ethernetType  ethernetII

port set $chas $card $rxPort

# Write config to hardware
ixWritePortsToHardware portList

# Clear stats, run the transmission
after 1000
ixClearPortStats $chas $card $rxPort
ixStartPortTransmit $chas $card $txPort
after 1000
ixCheckPortTransmitDone $chas $card $txPort

# Get the 8 QoS statistics and print them
stat get allStats $chas $rxPort

for {set i 0} {$i <= 7} {incr i} \
{
    ixPuts -nonewline "Qos$i = "
    ixPuts [stat cget -qualityOfService$i]
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

[stat](#), [port](#)

NAME - rip

rip - configure the RIP header parameters for a port on a card on a chassis

SYNOPSIS

rip *sub-command options*

DESCRIPTION

The **rip** command is used to configure the RIP header information used when building RIP-type packets. See RFCs 1058 and 1723 for a complete definition of RIP header fields. Note that *stream* get must be called before this command's get sub-command.

STANDARD OPTIONS

command

The command field of the RIP header. Defined values include:

Option	Value	Usage
<i>ripRequest</i>	1	(default) a request for the responding system to send all or part of its routing table
<i>ripResponse</i>	2	response or update information from a sender
<i>ripTraceOn</i>	3	an obsolete message
<i>ripTraceOff</i>	4	an obsolete message
<i>ripReserved</i>	5	reserved for use by Sun Microsystems

version

The version field of the RIP header. Defined values include:

Option	Value	Usage
<i>ripVersion1</i>	1	
<i>ripVersion2</i>	2	(default)

COMMANDS

The **rip** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

rip **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rip** command.

rip **config** *option value*

Modify the RIP configuration options of the port. If no *option* is specified, returns a list describing all of the available RIP options (see STANDARD OPTIONS) for port.

rip **decode** *capFrame [chassisID cardID portID]*

Decodes a captured frame in the capture buffer and updates TclHal. **rip cget** *option* command can be used after decoding to get the option data. Specific errors are:

- No connection to a chassis
- Invalid port number

- The captured frame is not a valid Rip frame

rip get *chasID cardID portID*

Gets the current RIP configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Note that *stream* get must be called before this command's get sub-command. Call this command before calling **rip cget option** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

rip set *chasID cardID portID*

Sets the RIP configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **rip config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

rip setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

# In this example we'll generate a RIP packet with two route
# specifications

# Connect to chassis and get chassis ID
set host 400-031561
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assume card to be used is in slot 1
set card 1
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
```

```
        return 1
    }
    # Take ownership of the ports we'll use
    if [ixTakeOwnership $portList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Put the port in loopback mode
    port setFactoryDefaults $chassis $card $port
    port setDefault

    # Stream: 1 packet at 1%
    stream setDefault
    stream config -numFrames      1
    stream config -dma           stopStream
    stream config -rateMode      usePercentRate
    stream config -percentPacketRate 1

    # Set up IP: udp with 72 byte packet
    ip setDefault
    ip config -ipProtocol          udp
    ip config -totalLength         72
    ip set $chassis $card $port

    # Set up protocol
    protocol setDefault
    protocol config -ethernetType   ethernetII
    protocol config -name           ipV4
    protocol config -appName        Rip

    # Set up UDP
    udp setDefault
    udp config -sourcePort        ripPort
    udp config -destPort          ripPort
    udp set $chassis $card $port

    # Set up Rip in general
    rip setDefault
    rip config -command            ripResponse
    rip config -version             2
    # Set up Rip Routes
    ripRoute setDefault
    ripRoute config -familyId       2
    ripRoute config -routeTag        0
    ripRoute config -metric          10
    ripRoute config -ipAddress       192.168.36.1
    ripRoute config -subnetMask      255.255.255.0
    ripRoute config -nextHop         192.168.46.254
    ripRoute set 1

    ripRoute config -metric          20
    ripRoute config -ipAddress       0.0.0.0
    ripRoute config -nextHop         192.168.46.1
    ripRoute set 2

    rip set $chassis $card $port

    stream set $chassis $card $port 1
    port set $chassis $card $port

    ixWritePortsToHardware portList
```

```
# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

port, protocol, ip, ripRoute

NAME - ripRoute

ripRoute - configure the RIP routing parameters for a port on a card on a chassis

SYNOPSIS	ripRoute <i>sub-command options</i>
DESCRIPTION	The ripRoute command is used to configure the RIP routing parameters used when building RIP packets. See RFCs 1058 and 1723 for a complete definition of RIP.
STANDARD OPTIONS	
authentication	Authentication string, maximum 16 octets. (<i>default</i> = "")
authenticationType	Type of authentication. (<i>default</i> = 2)
familyId	Address family identifier. Valid values are 2 (IP protocol), OxFFFF (authentication entry, automatically sets if ripRoute setAuthentication called). (<i>default</i> = 0)
ipAddress	IP address of the routing table entry. (<i>default</i> = 0.0.0.0)
metric	The routing cost metric, from 1 to 16 with 16 interpreted as unreachable. (<i>default</i> = 1)
nextHop	For version 2 records, the IP address of the next routing hop for the IP address and subnet mask. (<i>default</i> = 0.0.0.0)
routeTag	The number used to distinguish the source of routing destination. (<i>default</i> = 0)
subnetMask	Subnet mask for this route. (<i>default</i> = 0.0.0.0)
COMMANDS	The ripRoute command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
ripRoute cget <i>option</i>	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the ripRoute command.
ripRoute config <i>option value</i>	Modify the ripRoute configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available ripRoute options (see STANDARD OPTIONS) for port.
ripRoute get <i>routeID</i>	Gets the current route configuration of the selected <i>routeID</i> . Call this command before calling ripRoute cget <i>option</i> to get the value of the configuration option. Specific errors are:

A

ripRoute

- The specified route does not exist

ripRoute remove *routeID*

Remove the route *routeID* from the routing table. Specific errors are:

- The specified route does not exist

ripRoute set *routeID*

Sets the route configuration for route *routeID* reading the configuration option values set by the **ripRoute config option value** command. Specific errors are:

- The configured parameters are not valid for this port
- Insufficient memory to add the new route

ripRoute setAuthentication *authentication*

Sets an authentication route as the first entry of the routing table with *familyID* set to 0xFFFF. Specific errors are:

- The parent rip structure does not exist
- Insufficient memory

ripRoute setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *rip*

SEE ALSO

stream, protocol, ip, rip

NAME - rprFairness

rprFairness - set up transmission of RPR fairness packets

SYNOPSIS

rprFairness *sub-command options*

DESCRIPTION

The **rprFairness** command is used to set up the content of RPR Fairness messages sent periodically from a node. The RPR Fairness Algorithm (FA) is used to manage congestion on the ringlets in an RPR network. Fairness frames are sent periodically to advertise bandwidth usage parameters to other nodes in the network to maintain weighted fair share distribution of bandwidth. The messages are sent in the direction opposite to the data flow; that is, on the other ringlet.

The Sonet header must be set to *sonetRpr* using *sonet config -header sonetRpr* before this command can be used.

STANDARD OPTIONS

RPR Fairness Options

controlValue

The 16-bit normalized advertised fair rate. A full rate is indicated by all 1's (0xFFFF). (*default = 0*)

enableTransmit true / false

If *true*, the transmission of RPR Fairness Control Messages (FCMs) is enabled. They are sent at the repeat interval specified in *repeatInterval* until this option is set to *false*. (*default = false*)

messageType

The type of RPR fairness control message (FCM) used for congestion control.

Option	Value	Usage
rprSingleChoke	0	(<i>default</i>) Single choke: sent once per advertisement interval. Contains information on the congestion level for the ringlet.
rprMultiChoke	1	Multi choke: sent once every 10 advertisement intervals.

repeatInterval

The time interval, expressed in microseconds, between transmissions of Fairness frames. The range is from 10 to 65,000 microseconds. (*default = 90*)

rxAgingInterval

The keepalive timeout value, expressed in microseconds, indicating the amount of time that may elapse without an RPR message being received before considering the link down. (*default = 100*)

rxMacAddress

The 6-byte MAC address from which the packet was sent. (*default = {00 00 00 00 00 00}*)

txMacAddress

The 6-byte MAC Source address for the transmitting node. (*default = {00 00 00 00 00 00}*)

RPR Ring Control Options

enableFairnessEligible
true / false

This 1-bit field indicates the eligibility of this packet for the fairness algorithm. Note that packets with *serviceClass* set to *rprServiceClassA0* or *rprServiceClassA1* are not eligible for fairness control. (*default = true*)

enableOddParity
true / false

If *true*, then the value of the transmitted parity is odd over the first two bytes (TTL and baseRingControl). If *false*, even parity is set. (*default = true*)

enableWrapEligible
true / false

This 1-bit field indicates whether the packet is wrap eligible. (*default = false*)

packetType

This 2-bit field indicates the type of the RPR packet.

Option	Value	Usage
rprIdlePacket	0	Idle frame.
rprControlPacket	1	Control frame, expect for Fairness frames.
rprRingControlPacket	2	(<i>default</i>) Fairness frame.
rprDataPacket	3	Data frame.

parityBit

Read only. The value of the parity associated with the ring control header. For use in RPR Fairness Frames only. The value of this field is influenced by the value of the *enableOddParity* field.

ringIdentifier

This 1-bit field is the Ringlet Identifier (RI), indicating the ringlet from which the RPR frame was first transmitted.

Option	Value	Usage
rprRinglet0	0	(<i>default</i>) Ringlet 0.
rprRinglet1	1	Ringlet 1.

serviceClass

This 2-bit field indicates the MAC service class for the frame.

Option	Value	Usage
rprServiceClassC	0	(<i>default</i>) Class C is the lowest level of traffic, transmitted on a best-efforts basis. None of the traffic has a guaranteed data rate, and no limits are placed on delay and jitter. ClassC traffic is eligible for use by the fairness algorithm.
rprServiceClassB	1	Class B is the next higher service level, with an allocated and guaranteed data rate for a portion of the traffic, plus low delay and jitter (CIR). The additional traffic is transmitted with no guaranteed data rate (EIR) and is eligible for use by the fairness algorithm.
rprServiceClassA1	2	Class A is the highest service level, providing an allocated and guaranteed data rate, plus low delay and jitter (CIR). It is not eligible for use by the fairness algorithm. There are two sub-classes, which are not distinguished to the MAC client: Class A1 and Class A0. Class A1 reserved bandwidth may be used by ClassB or ClassC traffic if not in current use.
rprServiceClassA0	3	Class A0 may not be reused if not in current use.

ttl

This 8-bit field indicates the Time to Live option of the RPR header. The TTL value is the first octet of an RPR frame header. This indicates the maximum number of hops to the destination. (*default = 1*)

COMMANDS

The **rprFairness** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

rprFairness cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprFairness** command.

rprFairness config *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

rprFairness get *chassisID cardID portID*

Gets the current configuration of the port with id *portID* on card *cardID*, chassis *chassisID* from its hardware. Call this command before calling **rprFairness cget *option value*** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- RPR is not supported on this port

rprFairness set *chassisID cardID portID*

Sets the configuration of the port in IxTclHAL with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **rprFairness config *option value*** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The port is not available
- The configured parameters are not valid for this port
- RPR is not supported on this port

rprFairness setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package req IxTclHal

set host      localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
```

```

if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chassis [ixGetChassisID $host]

set chassis [chassis cget -id]
set card    87
set port    1
set portList [list [list $chassis $card $port] ]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# NOTE: Sonet header needs to be configured to sonetRpr before
# the user can configure any RPR streams

# Make sure we have all the default with sonetRpr before
# configuring anything on the port
if [port isValidFeature $chassis $card $port $::portFeatureRpr] {
    port setFactoryDefaults $chassis $card $port

    if [sonet get $chassis $card $port] {
        ixPuts "Error getting sonet on $chassis $card $port"
        return $TCL_ERROR
    }
    sonet config -header sonetRpr

    if [sonet set $chassis $card $port] {
        ixPuts "Error setting sonet on $chassis $card $port"
        return $TCL_ERROR
    }
} else {
    ixPuts "Port $chassis.$card.$port doesn't support RPR"
    return $TCL_ERROR
}

ixWritePortsToHardware portList

stream setDefault
stream config -rateMode          usePercentRate
stream config -framesize         1000
stream config -frameSizeType    sizeFixed

rprFairness setDefault
rprFairness config -repeatInterval      10
rprFairness config -packetType          rprFairnessPacket
rprFairness config -serviceClass        rprServiceClassB
rprFairness config -enableWrapEligible true
rprFairness config -enableOddParity    true

if [rprFairness set $chassis $card $port] {
    ixPuts "Error setting rprFairness on $chassis $card $port"
    return $TCL_ERROR
}

```

```

}

set sn 1

#####
# RPR Topology #####
# NOTE: protocol need to be configured before rprRingControl or
# any other RPR objects

protocol setDefault
protocol config -appName RprTopology

rprRingControl setDefault
rprRingControl config -ttl      5
rprRingControl config -extendedFrame    1

if [rprRingControl set $chassis $card $port] {
    ixPuts "Error setting rprRingControl on $chassis $card $port"
    return $TCL_ERROR
}

rprTopology clearAllTlvs

rprTlvIndividualBandwidth    clearAllBandwidthPairs
rprTlvBandwidthPair setDefault
rprTlvBandwidthPair config -bandwidth0 11
rprTlvBandwidthPair config -bandwidth1 11
rprTlvIndividualBandwidth    addBandwidthPair

rprTlvBandwidthPair setDefault
rprTlvBandwidthPair config -bandwidth0 22
rprTlvBandwidthPair config -bandwidth1 22
rprTlvIndividualBandwidth    addBandwidthPair

rprTlvIndividualBandwidth    setDefault
rprTopology addTlv rprIndividualBandwidth

rprTlvWeight setDefault
rprTlvWeight config -weightRinglet0    1
rprTlvWeight config -weightRinglet1    1
rprTopology addTlv rprWeight

rprTlvTotalBandwidth setDefault
rprTlvTotalBandwidth config -bandwidthRinglet0 1
rprTlvTotalBandwidth config -bandwidthRinglet1 1
rprTopology addTlv rprTotalBandwidth

rprTlvNeighborAddress setDefault
rprTlvNeighborAddress config -neighborMacEast {00 00 00 00 00
01}
rprTlvNeighborAddress config -neighborMacWest {00 00 00 00 00
02}
rprTopology addTlv rprNeighborAddress

rprTlvStationName setDefault
rprTlvStationName config -stationName newyorkcity
rprTopology addTlv rprStationName

rprTlvVendorSpecific setDefault
rprTlvVendorSpecific config -companyId {99 AA BB}
rprTlvVendorSpecific config -dependentId {23 45 67}
rprTlvVendorSpecific config -vendorData {11 11 11 10}

```

```

rprTopology addTlv rprVendorSpecific

rprTopology setDefault
rprTopology config -enableOverrideControlVersion $::false
rprTopology config -controlVersion          0
rprTopology config -enableOverrideControlType $::false
rprTopology config -controlType            1

if [rprTopology set $chassis $card $port] {
    ixPuts "Error setting rprTopology on $chassis $card $port"
    return $TCL_ERROR
}

stream config -name "RPR Topology"
if [stream set $chassis $card $port $sn] {
    ixPuts "Error setting stream on $chassis $card $port $sn"
    return $TCL_ERROR
}
incr sn

##### RPR Protection #####
protocol config -appName RprProtection

rprRingControl setDefault
rprRingControl config -ttl      5
rprRingControl config -ttlBase  6

if [rprRingControl set $chassis $card $port] {
    ixPuts "Error setting rprRingControl on $chassis $card $port"
    return $TCL_ERROR
}

rprProtection setDefault
rprProtection config -sequenceNumber      1
rprProtection config -protectionRequestEast rprWaitToRestore
rprProtection config -protectionRequestWest rprWaitToRestore
rprProtection config -enableOverrideControlType $::false
rprProtection config -controlType          2

if [rprProtection set $chassis $card $port] {
    ixPuts "Error setting rprProtection on $chassis $card $port"
    return $TCL_ERROR
}
stream config -name "RPR Protection"

if [stream set $chassis $card $port $sn] {
    ixPuts "Error setting stream on $chassis $card $port $sn"
    return $TCL_ERROR
}
incr sn

##### RPR OAM #####
protocol config -appName RprOam
rprOam setDefault
rprOam config -typeCode      $::rprOamFlush
rprOam config -flushReserved 18

if [rprOam set $chassis $card $port] {
    ixPuts "Error setting rprOam on $chassis $card $port"
    return $TCL_ERROR
}

```

```

stream config -name "RPR OAM"

if [stream set $chassis $card $port $sn] {
    ixPuts "Error setting stream on $chassis $card $port $sn"
    return $TCL_ERROR
}
incr sn

##### RPR TCP/IP #####
protocol config -name          ipV4
protocol config -appName        0
protocol config -ethernetType  noType

ip setDefault
ip config -ipProtocol    tcp

if [ip set $chassis $card $port] {
    ixPuts "Error ip stream on $chassis $card $port"
    return $TCL_ERROR
}
stream config -name "RPR TCP/IP"

if [stream set $chassis $card $port $sn] {
    ixPuts "Error setting stream on $chassis $card $port $sn"
    return $TCL_ERROR
}
incr sn

##### RPR ARP #####
protocol setDefault
protocol config -name          mac
protocol config -appName        Arp
protocol config -ethernetType  noType

rprRingControl setDefault
rprRingControl config -enableWrapEligible      $::true
rprRingControl config -enableOddParity         $::false

if {[rprRingControl set $chassis $card $port]} {
    ixPuts "Error setting rprRingControl on $chassis $card $port"
    return $TCL_ERROR
}

arp setDefault
arp config -sourceProtocolAddr 9.9.9.3
arp config -destProtocolAddr   8.8.8.3

if [arp set $chassis $card $port] {
    ixPuts "Error setting arp on $chassis $card $port"
    return $TCL_ERROR
}
stream config -name "RPR ARP"
if [stream set $chassis $card $port $sn] {
    ixPuts "Error setting stream on $chassis $card $port $sn"
    return $TCL_ERROR
}

ixWriteConfigToHardware portList

# Post transmission.
stat get statAllStats $chassis $card $port

```

```
stat cget -rprFairnessFramesReceived
stat getRate statRprPayloadCrcErrors $chassis $card $port

#
# Managing the Tlvs
#

set tlvObjectPointer [rprTopology getFirstTlv]
set tlvType [$tlvObjectPointer cget -type]

showCmd $tlvObjectPointer

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

*rprOam, rprProtection, rprRingControl, rprTlvBandwidthPair,
rprTlvIndividualBandwidth, rprTlvNeighborAddress, rprTlvStationName,
rprTlvTotalBandwidth, rprTlvVendorSpecific, rprTlvWeight, rprTopology*

NAME - rprOam

rprOam - build RPR OAM messages

SYNOPSIS

rprOam *sub-command options*

DESCRIPTION

The **rprOam** command is used to build RPR OAM (Operations, Administration, Management) messages. These messages are sent between stations to determine the operational status of the connection. There are three types of messages:

- Echo request and response frames: to determine connectivity.
- Flush frames: to prevent mis-ordering of frames.
- Vendor specific frames: for carrying a vendor's OAM information.

STANDARD OPTIONS

flushReserved

A 4-bit field reserved for future use, to be used only when *typeCode* is *rprOamFlush*. (*default* = 0)

requestProtectionMode

Indicates the protection mode of the request; used by the receiving station to determine which ringlet to respond on.

Option	Value	Usage
rprOamProtected	0	Protected.
rprOamUnProtected	1	(<i>default</i>) Not protected.

requestReserved

Read-only. A 4-bit field reserved for future use.

requestRinglet

The requested response ringlet.

Option	Value	Usage
rprOamReplyOnDefault	0	Reply using the default calculation.
rprOamReplyOnRinglet0	1	Reply on ringlet 0.
rprOamReplyOnRinglet1	2	(<i>default</i>) Reply on ringlet 1.
rprOamReplyReserved	3	Reserved for future use.

responseProtectionMode

Holds the same value of the *requestProtectionMode* for a received echo request. (*default* = *rprOamUnProtected*)

responseReserved

Read-only. A 4-bit field reserved for future use.

responseRinglet

Holds the same value of the *requestRinglet* for a received echo request. (*default* = \$::rprOamReplyOnRinglet1)

typeCode

The OAM type code for the message.

Option	Value	Usage
rprOamFlush	1	Flush message.
rprOamEchoRequest	8	Echo request message.
rprOamEchoResponse	9	(<i>default</i>) Echo response message.

Option	Value	Usage
rprOamVendorSpecific	15	Vendor specific message.

typeReserved

Read-only. A 4-bit field reserved for future use.

vendorOui

This option is used only when *typeCode* has a value of *rprOamVendorSpecific* and is the 3-octet IEEE company identifier for this vendor. The user data for this command may be set in the stream's background data. (*default* = {55 55 77})

COMMANDS

The **rprOam** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

rprOam cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprRingControl** command.

rprOam config option value

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

rprOam decode capSlice chassisID cardID portID

Decodes a slice/frame into the **rprOam** variables. If not an **rprOam** frame, returns TCL_ERROR. May be used to determine if the captured frame is a valid **rprOam** frame. Specific errors are:

- No connection to a chassis
- RPR is not a supported feature on this port

rprOam get chassisID cardID portID

Gets the current configuration of the port with id *portID* on card *cardID*, chassis *chassisID* from its hardware. This call must have been preceded by a call to *rprOam set* or *stream get*. Call this command before calling **rprOam cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- Data not available, do a *stream get*

rprOam set chassisID cardID portID

Sets the configuration of the port in IxTclHAL with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **rprOam config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- RPR is not a supported feature on this port

rprOam setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under [*rprFairness*](#).

SEE ALSO

[*rprFairness*](#), [*rprProtection*](#), [*rprRingControl*](#), [*rprTlvBandwidthPair*](#),
[*rprTlvIndividualBandwidth*](#), [*rprTlvNeighborAddress*](#), [*rprTlvStationName*](#),
[*rprTlvTotalBandwidth*](#), [*rprTlvVendorSpecific*](#), [*rprTlvWeight*](#), [*rprTopology*](#)

NAME - rprProtection

rprProtection - build RPR protection messages

SYNOPSIS	rprProtection <i>sub-command options</i>
-----------------	--

DESCRIPTION	The rprProtection command is used to build RPR protection messages. Protection messages provide wrapping status information and indicates of a station's desires with respect to wrapping.
--------------------	---

STANDARD OPTIONS

controlType If *enableOverrideControlType* is set to *true*, then this is the new control type value to be set in the message.

Option	Value	Usage
rprDiscovery	1	Discovery message.
rprProtection	2	Protection message.
rprOamControl	3	(default) OAM message.

controlVersion If *enableOverrideControlVersion* is set to *true*, then this is the new control version value to be set in the message. (*default* = 0)

enableOverrideControlType *true* / *false* The message control type is normally set appropriately for the type of message being formatted. Setting this option to *true*, allows that type setting to be changed as specified in *controlType*. (*default* = *false*)

enableOverrideControlVersion *true* / *false* The message control version is normally set to 0. Setting this option to *true*, allows that version setting to be changed as specified in *controlVersion*. (*default* = *false*)

headerChecksum *Read-only*. The 16-bit header error (hec) checksum calculated over the control header.

jumboPreferred Indicates a station's ability and/or preference to support jumbo frames. A *false* value indicates that the station cannot support jumbo frames or prefers not to do so. A *true* value indicates that the station can support jumbo frames and prefers to do so. (*default* = *false*)

protectionRequestEast The RPR protection message type to report the protection state on the east interface of this station.

Option	Value	Usage
rprNoRequest	0	(default) No requested type.
rprWaitToRestore	1	Wait to restore.
rprManualSwitch	2	Specifies that the indicated link should not be used.
rprSignalDegrade	3	A minor signal degradation condition exists.
rprSignalFair	4	A major signal degradation condition exists and the link may not be used.

Option	Value	Usage
rprForcedSwitch	5	Specifies that the indicated link may not be used.

protectionRequestWest

The RPR protection message type to report the protection state on the west interface of this station. See *protectionRequestEast* for a list of choices and the default value.

sequenceNumber

This 8-bit field has a valid range of 0 to 63. This field is the sequence number used with all copies of a particular protection control message. The value is incremented only if the contents of the message packet change, ensuring that protection control messages are processed in the correct order. (*default = 0*)

wrapPreferred

Indicates a station's ability and/or preference to support wrapping protection. A *false* value indicates that the station cannot support wrap protection or prefers not to do so. A *true* value indicates that the station can support wrap protection and prefers to do so. (*default = 0*)

wrappingStatusEast

The wrapping status for the traffic received on the east interface of this station. A *true* value indicates that the traffic is wrapped, and a *false* value indicates that the traffic is not enabled. (*default = false*)

wrappingStatusWest

The wrapping status for the traffic received on the west interface of this station. A *true* value indicates that the traffic is wrapped, and a *false* value indicates that the traffic is not enabled. (*default = false*)

COMMANDS

The **rprProtection** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

rprProtection cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprRingControl** command.

rprProtection config option value

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

rprProtection decode capSlice chassisID cardID portID

Decodes a slice/frame into the **rprProtection** variables. If not an **rprProtection** frame, returns TCL_ERROR. May be used to determine if the captured frame is a valid **rprProtection** frame. Specific errors are:

- Invalid port
- No connection to a chassis
- The captured frame is not an *rprProtection* frame
- RPR is not a supported feature on this port

rprProtection get chassisID cardID portID

Gets the current configuration of the port with id *portID* on card *cardID*, chassis *chassisID* from its hardware. This call must have been preceded by a call to

rprProtection set or *stream get*. Call this command before calling **rprProtection cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- Data not available, do a *stream get*

rprProtection set *chasID cardID portID*

Sets the configuration of the port in IxTclHAL with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **rprProtection config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- RPR is not a supported feature on this port

rprProtection setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *rprFairness*.

SEE ALSO

rprFairness, rprOam, rprRingControl, rprTlvBandwidthPair, rprTlvIndividualBandwidth, rprTlvNeighborAddress, rprTlvStationName, rprTlvTotalBandwidth, rprTlvVendorSpecific, rprTlvWeight, rprTopology

NAME - rprRingControl

rprRingControl - set up Ring Control header for RPR packets

SYNOPSIS

rprRingControl *sub-command options*

DESCRIPTION

The **rprRingControl** command is used to set up the content of RPR header used by all RPR packets except the RPR Fairness Frames, which are set up in the *rprFairness* command. The options are divided into Base Control and Extended Control options.

STANDARD OPTIONS

RPR Base Control Options

enableFairnessEligible *true / false*

This 1-bit field indicates the eligibility of this packet for the fairness algorithm. Note that packets with *serviceClass* set to *rprServiceClassA0* or *rprServiceClassA1* are not eligible for fairness control. (*default = true*)

enableOddParity *true / false*

If *true*, then the value of the transmitted parity is odd over the first two bytes (TTL and baseRingControl). If *false*, even parity is set. (*default = true*)

enableWrapEligible *true / false*

This 1-bit field indicates whether the packet is wrap eligible. (*default = false*)

packetType

This 2-bit field indicates the type of the RPR packet.

Option	Value	Usage
rprControlPacket	1	Control frame, expect for Fairness frames.
rprDataPacket	3	(<i>default</i>) Data frame.

parityBit

Read only. The value of the parity associated with the ring control header. For use in RPR Fairness Frames only. The value of this field is influenced by the value of the *enableOddParity* field.

ringIdentifier

This 1-bit field is the Ringlet Identifier (RI), indicating the ringlet from which the RPR frame was first transmitted.

Option	Value	Usage
rprRinglet0	0	(<i>default</i>) Ringlet 0.
rprRinglet1	1	Ringlet 1.

serviceClass

This 2-bit field indicates the MAC service class for the frame.

Option	Value	Usage
rprServiceClassC	0	Class C is the lowest level of traffic, transmitted on a best-efforts basis. None of the traffic has a guaranteed data rate, and no limits are placed on delay and jitter. ClassC traffic is eligible for use by the fairness algorithm.

Option	Value	Usage
rprServiceClassB	1	(<i>default</i>) Class B is the next higher service level, with an allocated and guaranteed data rate for a portion of the traffic, plus low delay and jitter (CIR). The additional traffic is transmitted with no guaranteed data rate (EIR) and is eligible for use by the fairness algorithm.
rprServiceClassA1	2	Class A is the highest service level, providing an allocated and guaranteed data rate, plus low delay and jitter (CIR). It is not eligible for use by the fairness algorithm. There are two sub-classes, which are not distinguished to the MAC client: Class A1 and Class A0. Class A1 reserved bandwidth may be used by ClassB or ClassC traffic if not in current use.
rprServiceClassA0	3	Class A0 may not be reused if not in current use.

ttl

This 8-bit field indicates the Time to Live option of the RPR header. The TTL value is the first octet of an RPR frame header. This indicates the maximum number of hops to the destination. (*default = 1*)

RPR Extended Control Options**extendedFrame**

This 1-bit field indicates that this data frame is sent from a MAC source which is not a node on the ring to a MAC destination that is not a node on the ring. If set to *true*, then the entire MAC layer packet is expected after the *hec* field in the RPR packet, including the destination and source MAC addresses. (*default = 0*)

floodingForm

This 2-bit field indicates whether the packet should be flooded and whether it should be flooded uni-directionally or bi-directionally.

Option	Value	Usage
rprFfNoFlood	0	(<i>default</i>) No flooding.
rprFfUnidirectionalFlood	1	Flood only in the ringlet specified in <i>ringIdentifier</i> .
rprFfBidirectionalFlood	2	Flood to both ringlets.
rprFfReserved	3	Reserved

passedSource

This 1-bit field is used by wrapping systems to prevent frame mis-order and duplication. It is normally set to 0 when a frame is first transmitted by a station and set to 1 when a wrapped frame passes the source station again. (*default = 0*)

reserved

A 3-bit reserved field for future use. It is normally set to 0's on transmission and ignored upon receipt. (*default = 0*)

strictOrder

This 1-bit field indicates whether strict ordering (1) or relaxed ordering (0) requirements should be observed. (*default = 0*)

ttlBase

If the value of *packetType* is *rprDataPacket*, then this 8-bit field should be set the original TTL of the data packet before RPR encapsulation. (*default = 0*)

COMMANDS

The **rprRingControl** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

rprRingControl cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprRingControl** command.

rprRingControl config *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

rprRingControl decode *capSlice chasID cardID portID*

Decodes a slice/frame into the **rprRingControl** variables. If not an **rprRingControl** frame, returns TCL_ERROR. May be used to determine if the captured frame is a valid **rprRingControl** frame. Specific errors are:

- No connection to a chassis
- The captured frame is not an *rprRingControl* frame
- RPR is not a supported feature on this port

rprRingControl get *chasID cardID portID*

Gets the current configuration of the port with id *portID* on card *cardID*, chassis *chasID* from its hardware. This call must have been preceded by a call to *rprRingControl set* or *stream get*. Call this command before calling **rprRingControl cget *option value*** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- Data not available, do a *stream get*

rprRingControl set *chasID cardID portID*

Sets the configuration of the port in IxTclHAL with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **rprRingControl config *option value*** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- RPR is not a supported feature on this port

rprRingControl setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under [rprFairness](#).

SEE ALSO

[rprFairness](#), [rprOam](#), [rprProtection](#), [rprTlvBandwidthPair](#),
[rprTlvIndividualBandwidth](#), [rprTlvNeighborAddress](#), [rprTlvStationName](#),
[rprTlvTotalBandwidth](#), [rprTlvVendorSpecific](#), [rprTlvWeight](#), [rprTopology](#)

NAME - rprTlvBandwidthPair

rprTlvBandwidthPair - defines a pair of bandwidth values for use in an RPR Individual Bandwidth TLV

SYNOPSIS	<i>rprTlvBandwidthPair sub-command options</i>
-----------------	--

DESCRIPTION	The rprTlvBandwidthPair command is used to set up a pair of bandwidth values. This pair is added to a RPR Individual Bandwidth TLV by use of the <i>rprTlvIndividualBandwidth addBandwidthPair</i> command. A bandwidth pair may be retrieved from the individual bandwidth list by calling <i>rprTlvIndividualBandwidth getFirstBandwidthPair/getNextBandwidthPair</i> and then inspecting the options in this command.
--------------------	--

STANDARD OPTIONS

bandwidth0	The bandwidth requirement associated with Ringlet 0. (<i>default = 0</i>)
bandwidth1	The bandwidth requirement associated with Ringlet 1. (<i>default = 0</i>)

COMMANDS	The rprTlvBandwidthPair command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands.
-----------------	--

rprTlvBandwidthPair **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprTlvBandwidthPair** command.

rprTlvBandwidthPair **config** *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

rprTlvBandwidthPair **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES	See examples under <i>rprFairness</i> .
-----------------	---

SEE ALSO	<i>rprFairness, rprOam, rprProtection, rprRingControl, rprTlvIndividualBandwidth, rprTlvNeighborAddress, rprTlvStationName, rprTlvTotalBandwidth, rprTlvVendorSpecific, rprTlvWeight, rprTopology</i>
-----------------	---

NAME - rprTlvIndividualBandwidth

rprTlvIndividualBandwidth - set up a TLV individual bandwidth item for use in an RPR topology message

SYNOPSIS

rprTlvIndividualBandwidth *sub-command options*

DESCRIPTION

The **rprTlvIndividualBandwidth** command is used to set up the content of an RPR Individual Bandwidth TLV for use in an RPR topology message. This TLV is added to a topology message by use of the *rprTopology addTlv rprIndividualBandwidth* command.

This command's data is constructed by adding *rprTlvBandwidthPairs*. Bandwidth pairs are constructed through the use of the *rprTlvBandwidthPair* command and then added to this command with the *rprTlvIndividualBandwidth addBandwidthPair* command. Each bandwidth pair corresponds to the reserved bandwidth between this node and a node a number of hops away from this node. The first item in the pair represents the reserved bandwidth on ringlet 0 and the second represents the reserved bandwidth on ringlet 1.

Bandwidth pairs must be added in order; that is, for the node one hop away, followed by the node two hops away, etc.

An individual bandwidth TLV may be retrieved from the topology TLV list by calling *rprTopology getFirstTlv/getNextTlv*, checking for *type = rprIndividualBandwidth* and then inspecting the options in this command.

STANDARD OPTIONS

TLV Common Options

dataLength

Read-only. The 10-bit length of the data fields.

reserved1

Read-only. The 6-bit Reserved1 field is set to 0 and ignored by receiving nodes.

reserved2

Read-only. The 6-bit Reserved2 field is set to 0 and ignored by receiving nodes.

type

Read-only. The 10-bit TLV type field, set to *rprIndividualBandwidth (3)*.

Individual Bandwidth TLV Specific Options

none

COMMANDS

The **rprTlvIndividualBandwidth** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands.

rprTlvIndividualBandwidth **addBandwidthPair**

Adds the bandwidth pair constructed in *rprTlvBandwidthPair* to the list of pairs. Specific errors include:

- Invalid parameters in *rprTlvBandwidthPair*

rprTlvIndividualBandwidth *cget* *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprTlvIndividualBandwidth** command.

rprTlvIndividualBandwidth *clearAllBandwidthPairs*

Clears all the bandwidth pairs.

rprTlvIndividualBandwidth *getFirstBandwidthPair*

Access the first bandwidth pair in the list. The pair's values may be read using the *rprTlvBandwidthPair* command. Specific errors are:

- There are no bandwidth pairs in the list

rprTlvIndividualBandwidth *getNextBandwidthPair*

Access the next bandwidth pair in the list. The pair's values may be read using the *rprTlvBandwidthPair* command. Specific errors are:

- There are no more bandwidth pairs in the list

rprTlvIndividualBandwidth *setDefault*

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *rprFairness*.

SEE ALSO

rprFairness, *rprOam*, *rprProtection*, *rprRingControl*, *rprTlvBandwidthPair*,
rprTlvNeighborAddress, *rprTlvStationName*, *rprTlvTotalBandwidth*,
rprTlvVendorSpecific, *rprTlvWeight*, *rprTopology*

NAME - rprTlvNeighborAddress

rprTlvNeighborAddress - set up a TLV neighbor address item for use in an RPR topology message

SYNOPSIS

rprTlvNeighborAddress *sub-command options*

DESCRIPTION

The **rprTlvNeighborAddress** command is used to set up the content of an RPR Neighbor Address TLV for use in an RPR topology message. This TLV is added to a topology message by use of the *rprTopology addTlv rprNeighborAddress* command.

A neighbor address TLV may be retrieved from the topology TLV list by calling *rprTopology getFirstTlv/getNextTlv*, checking for *type = rprNeighborAddress* and then inspecting the options in this command.

STANDARD OPTIONS

TLV Common Options

dataLength	<i>Read-only</i> . The 10-bit length of the data fields.
reserved1	<i>Read-only</i> . The 6-bit Reserved1 field is set to 0 and ignored by receiving nodes.
reserved2	<i>Read-only</i> . The 6-bit Reserved2 field is set to 0 and ignored by receiving nodes.
type	<i>Read-only</i> . The 10-bit TLV type field, set to <i>rprNeighborAddress</i> (4).

Neighbor Address TLV Specific Options

neighborMacEast	The 6-byte MAC address of the neighbor station connected to this station's east interface. This value is 0 when the MAC address is unknown. (<i>default = {00 00 00 00 00 00}</i>)
neighborMacWest	The 6-byte MAC address of the neighbor station connected to this station's west interface. This value is 0 when the MAC address is unknown. (<i>default = {00 00 00 00 00 00}</i>)

COMMANDS

The **rprTlvNeighborAddress** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands.

rprTlvNeighborAddress *cget option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprTlvNeighborAddress** command.

rprTlvNeighborAddress *config option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

rprTlvNeighborAddress **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *rprFairness*.

SEE ALSO

rprFairness, rprOam, rprProtection, rprRingControl, rprTlvBandwidthPair, rprTlvIndividualBandwidth, rprTlvStationName, rprTlvTotalBandwidth, rprTlvVendorSpecific, rprTlvWeight, rprTopology

NAME - rprTlvStationName

rprTlvStationName - set up a TLV station name item for use in an RPR topology message

SYNOPSIS

rprTlvStationName *sub-command options*

DESCRIPTION

The **rprTlvStationName** command is used to set up the content of an RPR Station Name TLV for use in an RPR topology message. This TLV is added to a topology message by use of the *rprTopology addTlv rprStationName* command. A station name TLV may be retrieved from the topology TLV list by calling *rprTopology getFirstTlv/getNextTlv*, checking for *type = rprStationName* and then inspecting the options in this command.

STANDARD OPTIONS

TLV Common Options

dataLength	<i>Read-only</i> . The 10-bit length of the data fields.
reserved1	<i>Read-only</i> . The 6-bit Reserved1 field is set to 0 and ignored by receiving nodes.
reserved2	<i>Read-only</i> . The 6-bit Reserved2 field is set to 0 and ignored by receiving nodes.
type	<i>Read-only</i> . The 10-bit TLV type field, set to <i>rprStationName</i> (5).

Station Name TLV Specific Options

stationName	The name of the station, expressed as a string. (<i>default = {}</i>)
--------------------	---

COMMANDS

The **rprTlvStationName** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

rprTlvStationName **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprTlvStationName** command.

rprTlvStationName **config** *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

rprTlvStationName **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *rprFairness*.

SEE ALSO

rprFairness, rprOam, rprProtection, rprRingControl, rprTlvBandwidthPair, rprTlvIndividualBandwidth, rprTlvNeighborAddress, rprTlvTotalBandwidth, rprTlvVendorSpecific, rprTlvWeight, rprTopology

NAME - rprTlvTotalBandwidth

rprTlvTotalBandwidth - set up a TLV total bandwidth item for use in an RPR topology message

SYNOPSIS	<i>rprTlvTotalBandwidth sub-command options</i>
-----------------	---

DESCRIPTION	The rprTlvTotalBandwidth command is used to set up the content of an RPR Total Bandwidth TLV for use in an RPR topology message. This TLV is added to a topology message by use of the <i>rprTopology addTlv rprTotalBandwidth</i> command. A total bandwidth TLV may be retrieved from the topology TLV list by calling <i>rprTopology getFirstTlv/getNextTlv</i> , checking for <i>type = rprTotalBandwidth</i> and then inspecting the options in this command.
--------------------	--

STANDARD OPTIONS

TLV Common Options

dataLength	<i>Read-only</i> . The 10-bit length of the data fields.
reserved1	<i>Read-only</i> . The 6-bit Reserved1 field is set to 0 and ignored by receiving nodes.
reserved2	<i>Read-only</i> . The 6-bit Reserved2 field is set to 0 and ignored by receiving nodes.
type	<i>Read-only</i> . The 10-bit TLV type field, set to <i>rprTotalBandwidth</i> (2).

Total Bandwidth TLV Specific Options

bandwidthRinglet0	The total reserved classA0 bandwidth value of the Ringlet 0 node for use in fairness calculations. (<i>default = 0</i>)
bandwidthRinglet1	The total reserved classA0 bandwidth value of the Ringlet 1 node for use in fairness calculations. (<i>default = 0</i>)

COMMANDS

The **rprTlvTotalBandwidth** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

rprTlvTotalBandwidth **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprTlvTotalBandwidth** command.

rprTlvTotalBandwidth **config** *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

rprTlvTotalBandwidth **setDefault**

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *rprFairness*.

SEE ALSO

rprFairness, rprOam, rprProtection, rprRingControl, rprTlvBandwidthPair, rprTlvIndividualBandwidth, rprTlvNeighborAddress, rprTlvStationName, rprTlvVendorSpecific, rprTlvWeight, rprTopology

NAME - rprTlvVendorSpecific

rprTlvVendorSpecific - set up a TLV Weight item for use in an RPR topology message

SYNOPSIS	rprTlvVendorSpecific <i>sub-command options</i>
-----------------	---

DESCRIPTION	<p>The rprTlvVendorSpecific command is used to set up the content of an RPR Vendor Specific TLV for use in an RPR topology message. This TLV is added to a topology message by use of the <i>rprTopology addTlv rprVendorSpecific</i> command.</p> <p>A vendor specific TLV may be retrieved from the topology TLV list by calling <i>rprTopology getFirstTlv/getNextTlv</i>, checking for <i>type = rprVendorSpecific</i> and then inspecting the options in this command.</p>
--------------------	--

STANDARD OPTIONS

TLV Common Options

dataLength	<i>Read-only</i> . The 10-bit length of the data fields.
reserved1	<i>Read-only</i> . The 6-bit Reserved1 field is set to 0 and ignored by receiving nodes.
reserved2	<i>Read-only</i> . The 6-bit Reserved2 field is set to 0 and ignored by receiving nodes.
type	<i>Read-only</i> . The 10-bit TLV type field, set to <i>rprVendorSpecific</i> (6).

Vendor Specific TLV Specific Options

companyId	A 3-byte hex value. This is the 24-bit IEEE/RAC company identifier, which is the first part of the globally unique EUI-64 identifier. (<i>default = {99 AA BB}</i>)
dependentId	A 3-byte hex value. This is the 24-bit identifier which is the second part of the globally unique EUI-64 identifier. This ID is supplied by the vendor and is unique to that vendor. (<i>default = {23 45 67}</i>)
vendorData	A variable amount of data specific to the <i>company</i> and <i>dependentId</i> . (<i>default = {}</i>)

COMMANDS	The rprTlvVendorSpecific command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	---

rprTlvVendorSpecific *cget option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprTlvVendorSpecific** command.

rprTlvVendorSpecific config option value

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

rprTlvVendorSpecific setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *rprFairness*.

SEE ALSO

rprFairness, rprOam, rprProtection, rprRingControl, rprTlvBandwidthPair, rprTlvIndividualBandwidth, rprTlvNeighborAddress, rprTlvStationName, rprTlvTotalBandwidth, rprTlvWeight, rprTopology

NAME - rprTlvWeight

rprTlvWeight - set up a TLV Weight item for use in an RPR topology message

SYNOPSIS	rprTlvWeight <i>sub-command options</i>
DESCRIPTION	<p>The rprTlvWeight command is used to set up the content of an RPR Weight TLV for use in an RPR topology message. This TLV is added to a topology message by use of the <i>rprTopology addTlv rprWeight</i> command.</p> <p>A weight TLV may be retrieved from the topology TLV list by calling <i>rprTopology getFirstTlv/getNextTlv</i>, checking for <i>type = rprWeight</i> and then inspecting the options in this command.</p>
STANDARD OPTIONS	
TLV Common Options	
dataLength	<i>Read-only</i> . The 10-bit length of the data fields.
reserved1	<i>Read-only</i> . The 6-bit Reserved1 field is set to 0 and ignored by receiving nodes.
reserved2	<i>Read-only</i> . The 6-bit Reserved2 field is set to 0 and ignored by receiving nodes.
type	<i>Read-only</i> . The 10-bit TLV type field, set to <i>rprWeight (1)</i> .
Weight TLV Specific Options	
weightRinglet0	The weight values of the Ringlet 0 node, to be used in fairness calculations. <i>(default = 0)</i>
weightRinglet1	The weight values of the Ringlet 1 node, to be used in fairness calculations. <i>(default = 0)</i>
COMMANDS	
The rprTlvWeight command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands.	
rprTlvWeight cget <i>option</i>	
Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the rprTlvWeight command.	
rprTlvWeight config <i>option value</i>	
Modify the configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If <i>option</i> is specified with no <i>value</i> , then the command returns a list of values available for this <i>option</i> .	
rprTlvWeight setDefault	
Sets to IxTclHal default values for all configuration options.	

EXAMPLES

See examples under *rprFairness*.

SEE ALSO

rprFairness, rprOam, rprProtection, rprRingControl, rprTlvBandwidthPair, rprTlvIndividualBandwidth, rprTlvNeighborAddress, rprTlvStationName, rprTlvTotalBandwidth, rprTlvVendorSpecific, rprTopology

NAME - rprTopology

rprTopology - build RPR topology messages

SYNOPSIS

rprTopology *sub-command options*

DESCRIPTION

The **rprTopology** command is used to build RPR topology messages. RPR topology messages consist of a set of TLV (type-length-value) settings constructed through the use of the *rprTlvIndividualBandwidth*, *rprTlvNeighborAddress*, *rprTlvStationName*, *rprTlvTotalBandwidth*, *rprTlvVendorSpecific*, and *rprTlvWeight* commands, followed by a call to the *addTlv* command for that type.

A TLV is added to a topology message by configuring the TLV with the appropriate command from the list above and then adding it to the topology message with *rprTopology addTlv* *type*, where *type* indicates which of the TLVs to use. A TLV may be retrieved from a topology message through the use of *getFirstTlv* / *getNextTlv*. These commands return the **name/pointer** of the command that was used to configure the TLV. This is typically used in the following sequence of commands:

```
set tlvcmd [rprTopology getFirstTlv]
$tlvcmd config ...
```

Each of the TLV commands also has a *type* option which uniquely identifies the type of the TLV.

STANDARD OPTIONS

controlType

If *enableOverrideControlType* is set to *true*, then this is the new control type value to be set in the message.

Option	Value	Usage
rprDiscovery	1	(default) Discovery message.
rprProtection	2	Protection message.
rprOamControl	3	OAM message.

controlVersion

If *enableOverrideControlVersion* is set to *true*, then this is the new control version value to be set in the message. (*default = 0*)

enableOverrideControl Type *true* / *false*

The message control type is normally set appropriately for the type of message being formatted. Setting this option to *true*, allows that type setting to be changed as specified in *controlType*. (*default = false*)

enableOverrideControl Version *true* / *false*

The message control version is normally set to 0. Setting this option to *true*, allows that version setting to be changed as specified in *controlVersion*. (*default = false*)

headerChecksum

Read-only. The 16-bit header error (hec) checksum calculated over the control header.

COMMANDS

The **rprTopology** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

rprTopology addTlv *tlvType*

Adds the TLV of type *tlvType* to the list. The choices of *tlvType* are:

Option	Value	Usage
rprWeight	1	TLV is built using the <i>rprTlvWeight</i> command.
rprTotalBandwidth	2	TLV is built using the <i>rprTlvTotalBandwidth</i> command.
rprIndividualBandwidth	3	TLV is built using the <i>rprTlvIndividualBandwidth</i> command.
rprNeighborAddress	4	TLV is built using the <i>rprTlvNeighborAddress</i> command.
rprStationName	5	TLV is built using the <i>rprTlvStationName</i> command.
rprVendorSpecific	6	TLV is built using the <i>rprTlvVendorSpecific</i> command.

Specific errors are:

- Invalid *tlvType*
- Invalid parameters in TLV

rprTopology cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **rprRingControl** command.

rprTopology clearAllTlvs

Clears all TLVs associated with the topology message.

rprTopology config *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

rprTopology decode *capSlice chassisID cardID portID*

Decodes a slice/frame into the **rprTopology** variables. If not an **rprTopology** frame, returns TCL_ERROR. May be used to determine if the captured frame is a valid **rprTopology** frame. This call also decodes each of the included TLVs in the slice/frame into the options associated with each of the separate TLV commands (*rprTlvIndividualBandwidth*, *rprTlvNeighborAddress*, *rprTlvStationName*, *rprTlvTotalBandwidth*, *rprTlvVendorSpecific* and *rprTlvWeight*). Specific errors are:

- Invalid port
- No connection to a chassis
- The captured frame is not an *rprTopology* frame
- RPR is not a supported feature on this port

rprTopology delTlv

Deletes the currently accessed TLV.

rprTopology get chasID cardID portID

Gets the current configuration of the port with id *portID* on card *cardID*, chassis *chasID* from its hardware. This call must have been preceded by a call to *rprTopology set* or *stream get*. Call this command before calling **rprTopology cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- Data not available, do a *stream get*

rprTopology getFirstTlv

Access the first TLV in the list. The results of the command is the **name** of the command used to make the TLV. This command may be symbolically used to view/modify the TLVs contents. Each TLV contains a *type* option that uniquely identifies the TLV's type. Specific errors are:

- There are no TLVs in the list

rprTopology getNextTlv

Access the next TLV in the list. The results of the command is the **name** of the command used to make the TLV. This command may be symbolically used to view/modify the TLVs contents. Each TLV contains a *type* option that uniquely identifies the TLV's type. Specific errors are:

- There are no more TLVs in the list

rprTopology set chasID cardID portID

Sets the configuration of the port in IxTclHAL with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **rprTopology config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- RPR is not a supported feature on this port

rprTopology setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *rprFairness*.

SEE ALSO

rprFairness, *rprOam*, *rprProtection*, *rprRingControl*, *rprTlvBandwidthPair*,
rprTlvIndividualBandwidth, *rprTlvNeighborAddress*, *rprTlvStationName*,
rprTlvTotalBandwidth, *rprTlvVendorSpecific*, *rprTlvWeight*

NAME - sequenceNumberUdf

sequenceNumberUdf - provides per-flow sequence numbers

SYNOPSIS	sequenceNumberUdf <i>sub-command options</i>
DESCRIPTION	<i>SequenceNumberUdf</i> is a new UDF field that provides per-flow sequence numbers. It knows what flow number is in a packet by pulling the flow number from an existing UDF 1-5. That existing UDF is known as the <i>associated UDF</i> . The feature is enabled with the <i>enable</i> option.
<hr/>	
STANDARD OPTIONS	
enable <i>true/false</i>	Enables/disables Sequence Number UDF function. (<i>default = false</i>)
byteOffset	The byte offset where the <i>sequenceNumberUdf</i> is placed. (<i>default = 0</i>)
associatedUdfID	The UDF that serves as the source for the flow number.(Note: The flow number is an input.) (<i>default = 1</i>)
associatedUdfBit Position	Bit offset of the flow number within the associated Udf. The value ranges from 0–7 for TPM cards. (<i>default = 0</i>)
associatedUdfWidth	Width to use for the associated Udf. The value ranges from 1–31 for TPM cards and 1–32 for all other card types.
associatedUdfMin	The minimum value of the associated Udf. Must be less than associatedUdfMax. (<i>default = 0</i>)
associatedUdfMax	The maximum value of the associated Udf. (<i>default = 4294967295</i>) Limitation: maximum - minimum < 2^{16} (48k for LM1000 series load modules) or maximum - minimum < 2^{19} (512k for LSM10G series load modules)
<hr/>	
COMMANDS	The sequenceNumberUdf command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
sequenceNumberUdf cget <i>option</i>	
Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the <i>sequenceNumberUdf</i> command.	
sequenceNumberUdf config <i>option value</i>	
Modify the sequenceNumberUdf options. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS).	

sequenceNumberUdf get *chasID cardID portID*

Gets the current configuration of the *sequenceNumberUdf* for port with id *portID* on card *cardID*, chassis *chasID* from its hardware. Note that *stream get* must be called before this sub-command. Call this command before calling *sequenceNumberUdf cget option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis.
- *stream get* has not been called.

sequenceNumberUdf set *chasID cardID portID*

Sets the configuration of the *sequenceNumberUdf* in IxHAL for a port by reading the configuration option values set by the *sequenceNumberUdf config option value* command. Specific errors are:

- No connection to a chassis
- Invalid port specification
- SequenceNumber UDFs are not supported on this port.
- The port is being used by another user

sequenceNumberUdf setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package req IxTclHal

set hostName woodstock

if {[ixConnectToChassis $hostName] } {
    Trace "Error connecting to $hostName"
}

set chassId      [chassis cget -id]
set portList [list]

set chassis      [chassis cget -id]
set card         4
set port         1
set streamId1

set portList [list [list $chassis $card $port]]

# Stream 1
stream setDefault
stream config -name           "Stream $streamId"
stream config -enable          $::true
stream config -framesize       100
stream config -frameSizeType   $::sizeFixed

udf setDefault
udf config -enable            $::true
udf config -offset             40
udf config -bitOffset          0
udf config -udfSize            8
udf config -initval            03
udf config -repeat              3
udf config -step                1
if {[udf set 1] } {
```

```
        errorMsg "Error setting udf 1."
    }

sequenceNumberUdf setDefault
sequenceNumberUdf config -enable $::true
sequenceNumberUdf config -byteOffset 42
sequenceNumberUdf config -associatedUdfID 1
sequenceNumberUdf config -associatedUdfBitPosition 0
sequenceNumberUdf config -associatedUdfWidth 4
sequenceNumberUdf config -associatedUdfMin 3
sequenceNumberUdf config -associatedUdfMax 5
if {[sequenceNumberUdf set $chassis $card $port]} {
    errorMsg "Error setting sequenceNumberUdf on $chassis
$card $port for stream $streamId."
}

if {[stream set $chassis $card $port $streamId]} {
    errorMsg "Error setting stream on $chassis $card $port for
stream $streamId."
}

ixWriteConfigToHardware portList
```

SEE ALSO[udf](#)

NAME - **serviceManager**

serviceManager - manage a multiuser session

SYNOPSIS

serviceManager *sub-command options*

DESCRIPTION

Most intelligent Ixia ports run the Linux Operating system. Software may be developed for these ports using the guidelines documented in the *Ixia Linux SDK Guide*. Such software must be combined in a set of files called a *package* and downloaded to a set of ports. Refer to [serviceManager](#) on page 5-124 for an overview of this command and details about package formats. Note this command is only valid in Windows based environments.

The *port* command's *isValidFeature* sub-command may be used to determine if a given port runs Linux. Use the following sequence:

```
if [port isValidFeature $chassis $card $port portFeatureIxRouter] {
    ... port runs Linux ...
}
```

STANDARD OPTIONS

none

COMMANDS

The **serviceManager** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

serviceManager deletePackage *packageList portGroupId*

Causes the packages included in *packageList* to be deleted from the ports included in *portGroupId*. The *packageList* is a space or comma separated list of package names. For example: '*package1,package2*'. The *portGroupId* is the ID used in the construction of a port group in the *portGroup create* command. Specific errors are:

- No connection to a chassis
- Invalid port list

serviceManager downloadPackage *packageList portGroupId*

Causes the packages included in *packageList* to be downloaded and started by the ports included in *portGroupId*. The *packageList* is a space or comma separated list of package names. For example: '*package1,package2*'. The *portGroupId* is the ID used in the construction of a port group in the *portGroup create* command. Specific errors are:

- No connection to a chassis
- Invalid port list
- One or more packages could not be found
- One or more packages were in improper format

serviceManager getInstalledPackages chassisID cardID portID

Returns a comma separated list of packages installed on the port. Specific errors are:

- No connection to a chassis
- Invalid port

EXAMPLES

```

package require IxTclHal
package require IxTclServices

set host localhost
set username user
# Assume card 1 is a card that supports Linux
set card 1
set port 1

# We'll use this port group
set portGroup 4242

# Package to be downloaded
set packageList [list "sample"]

# If we're on Unix, connect through Tcl Server
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get resulting ID
set chas [ixGetChassisID $host]

# Make sure that there's nothing in the port group
# Then put the single port in it
portGroup destroy $portGroup
if [portGroup create $portGroup] {
    ixPuts "Could not create port group"
    return 1
}
if [portGroup add $portGroup $chas $card $port] {
    ixPuts "Could not add port group"
    return 1
}

# Make sure that this port runs Linux
if {[port isValidFeature $chas $card $port \
    portFeatureIxRouter] == 0} {
    ixPuts "$chas:$card does not have a local CPU"
    return 1
}

# Download the package to the prt
if [serviceManager downloadPackage $packageList $portGroup] {

```

```
    ixPuts "Could not download package"
    return 1
}

# Check that it's there
ixPuts -none newline "Installed packages are: "
ixPuts [serviceManager getInstalledPackages $chas $card $port]

# And then remove it and check again
if [serviceManager deletePackage $packageList $portGroup] {
    ixPuts "Could not delete package"
    return 1
}

ixPuts -none newline "Installed packages are: "
ixPuts [serviceManager getInstalledPackages $chas $card $port]
```

SEE ALSO

ixLogin, *ixLogout*

NAME - session

session - manage a multiuser session

SYNOPSIS

session *sub-command options*

DESCRIPTION

The **session** command is used to login and logout of this TCL session. A user is not required to login to configure ports; however to take ownership of a group of ports in a multiuser environment, the user must log in. Session login is valid for the entire duration of a TCL window, regardless of how many times a *package require IxTclHal* or *cleanUp* is initiated or until the user logs out. Logging in as a different user name is the same as logging out and logging in again with a different login name.

STANDARD OPTIONS

captureBufferSegmentSize

Sets the size of the capture buffer request, in MB. The capture buffer is delivered in a series of segments that are no larger than this setting. (*default = 16*)

Note: *captureBufferSegmentSize* sets *this* client's request size, but does not affect any other client sessions.

userName

Read-only. User name for this session. (*default = “”*)

COMMANDS

The **session** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

session **config option value**

Modify the configuration options of the session. If no *option* is specified, returns a list describing all of the available **session** options (see STANDARD OPTIONS).

session **cget option**

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **session** command.

session **get option**

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **session** command.

session **login userName**

Initiate a login to a new multiuser session. If already logged in as a different *userName*, log out existing user and log in new *userName*. Specific errors are:

- No connection to a chassis
- *UserName* is null

session **logout**

Logs out current user.

session **set** *option*

Sets the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **session** command.

EXAMPLES

```
package require IxTclHal

# Login for george
session login george

# See who's logged in
set userName [session cget -userName]
ixPuts "$userName is currently logged in"

# Logout
session logout

# And check again
set userName [session cget -userName]
ixPuts "$userName is currently logged in"
```

SEE ALSO

ixLogin, *ixLogout*

NAME - sfpPlus

sfpPlus - configure the SFP+ transceiver interface.

SYNOPSIS

sfpPlus *sub-command options*

DESCRIPTION

The **sfpPlus** command is used to configure the SFP+ transceiver interface.

The small form-factor pluggable (SFP) transceiver interface capability has been added to NGY and other 10GE load modules.

STANDARD OPTIONS

enableMonitorLos <i>true/false</i>	Enable monitor SFP Loss of Signal. The interface requires the absence of a Loss of Signal for transmitting and receiving. (<i>default = false</i>)
enableMonitorModuleReadySignal <i>true/false</i>	Enable monitor SFP Module Ready Signal. The interface requires the detection of a Module Ready signal for transmitting and receiving. (<i>default = false</i>)
enableAutomaticDetect <i>true/false</i>	Enable automatic detection of transceiver type. (<i>default = false</i>)
type	Use to configure the transceiver type.

Option	Value	Usage
<i>sfpPlus10GBaseSrLr</i>	0	configure the transceiver to Limiting mode
<i>sfpPlus10GBaseLrm</i>	1	configure the transceiver to Linear mode
<i>sfpPlusCu</i>	2	configure the transceiver to Twinax (copper) mode
<i>sfpPlusCuPassive</i>	3	configure the transceiver to Passive (copper) mode
<i>sfpPlusNotDetected</i>	4	configure the transceiver to Not Detected (copper) mode

txPreTapControlValue Signifies the transmission of pre tap control value.

txMainTapControlValue Signifies the transmission of main tap control value.

txPostTapControlValue Signifies the transmission of post tap control value.

rxEqualizerControlValue Signifies the reception of equalizer control value.

COMMANDS

The **sfpPlus** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

sfpPlus config option value

Modify the configuration options. If no *option* is specified, returns a list describing all of the available **sfpPlus** options (see STANDARD OPTIONS).

sfpPlus cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **sfpPlus** command.

sfpPlus get chasID cardID portID

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **sfpPlus** command.

sfpPlus set chasID cardID portID

Sets the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **session** command.

sfpPlus setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

sfpPlus setDefault
sfpPlus config -enableMonitorLos false
sfpPlus config -enableMonitorModuleReadySignal $::false
sfpPlus config -enableAutomaticDetect $::false
sfpPlus config -type $::sfpPlus10GBaseLm
if {[sfpPlus set $chassis $card $port]} {
    errorMsg "Error calling sfpPlus set $chassis $card $port"
}
set pl [list [list $chassis $card $port]]
ixWritePortsToHardware portlist
```

SEE ALSO

NAME - sonet

sonet - configure the sonet properties of a POS port of a card on a chassis.

SYNOPSIS

sonet *sub-command options*

DESCRIPTION

The **sonet** command is used to configure the sonet properties of a POS port of a card on a chassis. **Note:** sonet error insertion is now handled by the *sonetError* command; sonet commands related to error insertion are now deprecated.

Note: the *setDefault* sub-command sets all options at default values, as indicated here. These values are a consistent setting for an OC12 card and may or may not be appropriate for other cards. In general, the sequence:

```
sonet setDefault
sonet set $chassis $card $port
```

fails.

The *port setFactoryDefaults* command, which relates to a particular port, sets all sonet options at default values appropriate for the type of port. The sequence:

```
port setFactoryDefaults $chassis $card $port
sonet set $chassis $card $port
```

always succeeds. If the use of *setFactoryDefaults* is undesirable, it is still essential that the value of *interfaceType* be set to a particular value after use of *setDefault*.

When the *headerType* is set to *sonetGfp*, the GFP header and overhead are set in the *gfp* and *gfpOverhead* commands.

STANDARD OPTIONS

apsType

Sets the Automatic Protection Switching (APS) bytes. Options include:

Option	Value	Usage
<i>linearAps</i>	0	(default) The K1 and K2 Automatic Protection Switching (APS) bytes bit definitions represent a linear topology.
<i>ringAps</i>	1	The K1 and K2 Automatic Protection Switching (APS) bytes bit definitions represent a ring topology.

C2byteExpected

Received path signal label. (*default* = 22)

C2byteTransmit

Register-programmable path signal label. (*default* = 22)

customK1K2 *true/false*

Enables or disables custom K1K2. (*default* = *false*)

dataScrambling *true/false*

Enables or disables data scrambling in the sonet framer. (*default* = *true*)

enableCiscoSrp

Enables the use of the particular packet formats for Cisco's implementation of SRP. *header* must be set to *sonetSrp* for this flag to have any effect. (*default* = *false*)

header

Enable sonet header type. Options include:

Option	Value	Usage
sonetHdlcPpplp	0	(default)
sonetCiscoHdlc	1	
sonetOther	2	
sonetFrameRelay1490	3	
sonetFrameRelay2427	3	
sonetFrameRelayCisco	4	
sonetSrp	5	not supported in channelized mode
sonetCiscoHdclpv6	6	
sonetHdlcPpplso	7	
sonetRpr	8	not supported in channelized mode
sonetAtm	9	
sonetGfp	10	Generic Framing Protocol.
sonetLaps	12	Link Access Procedure

interfaceType

Sets the type/speed of the sonet interface. Options include:

Option	Value	Usage
oc3	0	
oc12	1	(default)
oc48	2	
stm1c	3	
stm4c	4	
stm16c	5	
oc192	6	
stm64c	7	
ethOverSonet	8	
ethOverSdh	9	

k1NewState

Enables the K1 byte code value to be sent in the Sonet frame. (It is used by sonnet APS (automatic protection switching) to implement a bit-oriented protocol for critical switching operations). (default = 0)

k2NewState

Enables the K2 byte code value as in *k1NewState*. (default = 0)

**lineErrorHandling
*true/false***

Enables/disables line error handling on the sonet interface. (default = false)

**lineScrambling
*true/false***

Enables or disables line scrambling in the sonet framer. Applies only to the POS/sonet interface ports. (default = true)

operation

Sets up the sonet interface/operation either as normal mode or loopback mode. Options include:

Option	Value	Usage
sonetNormal	0	(default)
sonetLoopback	1	

Option	Value	Usage
sonetLineLoopback	2	
sonetFramerParallelDiagnosticLoopback	3	
sonetFramerDiagnosticLoopback	4	
sonetFecDiagnosticLoopback	5	
sonetFecLineLoopback	6	

pathErrorHandling
true/false

Enables or disables path error handling on the sonet interface. (*default = false*)

rprHecSeed

When RPR is used, this setting is used to indicate the HEC (Hardware Error Correction) seed value. Options include:

Option	Value	Usage
hecSeed)x0000	0	(<i>default</i>) 0 value
hecSeed0xffff	1	0xFFFF value.

rxCrc

Sets the receive CRC mode. Options include:

Option	Value	Usage
sonetCrc16	0	Selects reception with 16 bit CRC
sonetCrc32	1	(<i>default</i>) Selects reception with 32 bit CRC

trafficMap

Sets the Tcl hardware transmit mode. Options include:

Option	Value	Usage
sonetMapSpe	0	(<i>default</i>) SPE = Synchronous Payload Envelope packet streams
sonetMapDcc	1	DCC = Data Communications Channel packet flows

txCrc

Sets the transmit CRC mode. Options include:

Option	Value	Usage
sonetCrc16	0	Selects transmission with 16 bit CRC
sonetCrc32	1	(<i>default</i>) Selects transmission with 32 bit CRC

useRecoveredClock

Set the sonet framer to use no clock, the recovered clock or an external clock. Options include:

Option	Value	Usage
sonetNoClock	0	No clock is used.
sonetRecoveredClock	1	(<i>default</i>) Use the recovered clock.
sonetExternalClock	2	Use the external clock.

DEPRECATED STANDARD OPTIONS**B1 true/false**

B2 *true/false*

B3 *true/false*

errorDuration

insertBipErrors
true/false

lossOfFrame
true/false

lossOfSignal
true/false

periodicB1
true/false

periodicB2
true/false

periodicB3
true/false

periodicLossOfFrame
true/false

periodicLossOfSignal
true/false

COMMANDS

The **sonet** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

sonet cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **sonet** command.

sonet config *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port. If *option* is specified with no *value*, then the command returns a list of values available for this *option*.

sonet get *chasID cardID portID*

Gets the current configuration of the port with id *portID* on card *cardID*, chassis *chasID*. from its hardware. Call this command before calling **sonet cget** *option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

sonet set *chassisID cardID portID*

Sets the configuration of the port in IxTclHAL with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **sonet config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- The port is not a Packet over Sonet port or 10Gigabit WAN.

sonet setDefault

Sets to IxTclHal default values for all configuration options.

sonet write *chassisID cardID portID*

Writes or commits the changes in IxHAL to hardware for port *portID*, card *cardID*, chassis *chassisID*. Before using this command, use the *sonet set* command to configure the stream related options in IxHAL.

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that an OC48 POS card is in slot 18
set card 18
set portList [list [list $chas $card 1]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the type of card and check if it's the correct type
card get $chas $card
```

```
set type [card cget -type]
if {$type != $cardPosOc48} {
    ixPuts "Card $card is not an OC48c POS card ($type)"
    return 1
}

# Reset to the defaults and then set several values
sonet setDefault
sonet config -interfaceType oc48
sonet config -header sonetCiscoHdLC
sonet config -lineErrorHandling enable
sonet config -rxCrc sonetCrc16
sonet config -txCrc sonetCrc16

# Set the parameters
if [sonet set $chassis $card 1] {
    ixPuts "Sonet set failed on $chassis.$card.1"
    return 1
}

ixWriteConfigToHardware portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

card, port, sonetError, sonetOverhead, sonetCircuit

NAME - sonetCircuit

sonetCircuit - setup the circuits of a POS port of a card on a chassis.

SYNOPSIS

sonetCircuit *sub-command options*

DESCRIPTION

The **sonetCircuit** command is used to setup the circuits of a POS port of a card on a chassis.

STANDARD OPTIONS

txActiveTimeslotList	Configure the active tx members. (<i>default</i> = "")
rxActiveTimeslotList	Configure the active rx members. (<i>default</i> = "")
txIdleTimeslotList	Configure the idle tx members. (<i>default</i> = "")
rxIdleTimeslotList	Configure the idle rx members. (<i>default</i> = "")
name	Configure the circuit name. (<i>default</i> = "")
direction	Configure the circuit direction. (<i>default</i> = <i>circuitBidirectionSymmetrical</i>) Available options:

Option	Value	Usage
<i>circuitUnidirectionTx</i>	0	Uni-direction transmit
<i>circuitUnidirectionRx</i>	1	Uni-direction receive
<i>circuitBidirectionSymmetrical</i>	2	(<i>default</i>) Bi-direction symmetrical
<i>circuitBidirectionAsymmetrical</i>	3	Bi-direction asymmetrical

rxType

Configure the rx payload speed. (*default* = *circuitPayloadRateSTS1mv*)
Available options:

Option	Value	Usage
<i>circuitPayloadRateSTS1</i>	101	STS-1/VC-3
<i>circuitPayloadRateSTS3c</i>	102	STS-3c/VC-4
<i>circuitPayloadRateSTS12c</i>	103	STS-12c/VC-4-4c
<i>circuitPayloadRateSTS48c</i>	104	STS-48c/VC-4-16c
<i>circuitPayloadRateSTS1mv</i>	501	(<i>default</i>) STS-1-Xv / VC-3-Xv
<i>circuitPayloadRateSTS3cmv</i>	502	STS-3c-Xv / VC-4-Xv

txType

Configure the tx payload speed. (*default* = *circuitPayloadRateSTS1mv*)
Available options:

Option	Value	Usage
<i>circuitPayloadRateSTS1</i>	101	STS-1/VC-3
<i>circuitPayloadRateSTS3c</i>	102	STS-3c/VC-4
<i>circuitPayloadRateSTS12c</i>	103	STS-12c/VC-4-4c

Option	Value	Usage
<i>circuitPayloadRateSTS48c</i>	104	STS-48c/VC-4-16c
<i>circuitPayloadRateSTS1mv</i>	501	(default) STS-1-Xv / VC-3-Xv
<i>circuitPayloadRateSTS3cmv</i>	502	STS-3c-Xv / VC-4-Xv

enableTxLcas

Enable the Lcas on transmit side. (*default = FALSE*)

enableRxLcas

Enable the Lcas on receive side. (*default = FALSE*)

index

Read only. This parameter is used to view the circuit index assigned by hardware. (*default = 0*)

COMMANDS

The **sonetCircuit** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

sonetCircuit cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **sonetCircuit** command.

sonetCircuit config *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS).

sonetCircuit setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example on page A-667.

SEE ALSO

sonetCircuitList, *sonetCircuitProperties*

NAME - sonetCircuitList

sonetCircuitList - set up the circuits of a POS port of a card on a chassis.

SYNOPSIS	sonetCircuitList <i>sub-command options</i>
DESCRIPTION	The sonetCircuitList command is used to set up all the circuits of a POS port of a card on a chassis.
STANDARD OPTIONS	
numCircuits	<i>Read only</i> . This parameter is used to display the number of existing circuits in the circuit list. (<i>default =0</i>)
COMMANDS	The sonetCircuitList command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
sonetCircuitList add	Adds a new circuit and verifies that the circuit can be added. Specific errors are: <ul style="list-style-type: none"> • No connection to a chassis • Not a supported feature on this port • The port is being used by another user • Configured parameters are not valid for this setting
sonetCircuitList cget <i>option</i>	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the sonetCircuitList command.
sonetCircuitList clearAllCircuits	Remove all the circuits from the Sonet circuit list. Specific errors are: <ul style="list-style-type: none"> • No connection to a chassis • Not a supported feature on this port • The port is being used by another user • Configured parameters are not valid for this setting
sonetCircuitList config <i>option value</i>	Modify the configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.
sonetCircuitList del <i>circuitID</i>	Deletes the circuit with the given ID. Specific errors are: <ul style="list-style-type: none"> • No connection to a chassis • Not a supported feature on this port • The port is being used by another user • Configured parameters are not valid for this setting

sonetCircuitList get circuitID

Gets the existing circuit with the given circuit ID. Specific errors are:

- No connection to a chassis
- Not a supported feature on this port
- The port is being used by another user
- Configured parameters are not valid for this setting

sonetCircuitList getFirst

Gets the first circuit from the Sonet circuit list. Return values:

- No connection to a chassis
- Not a supported feature on this port
- The port is being used by another user
- Configured parameters are not valid for this setting

sonetCircuitList getNext

Gets the next circuit from the Sonet circuit list. Specific errors are:

- No connection to a chassis
- Not a supported feature on this port
- The port is being used by another user
- Configured parameters are not valid for this setting

sonetCircuitList select chasID cardID portID

Select the port where the IxTclHal configurations is set to local IxHal. Specific errors are:

- No connection to a chassis
- Invalid port number
- Not a supported feature on this port
- The port is being used by another user
- Configured parameters are not valid for this setting

sonetCircuitList set circuitID

Modify the existing circuit with the given circuit ID. Specific errors are:

- No connection to a chassis
- Not a supported feature on this port
- The port is being used by another user
- Configured parameters are not valid for this setting

sonetCircuitList setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package req IxTclHal

set hostname1600T-2

if {[ixConnectToChassis $hostname]} {
    errorMsg "error connecting $hostname chassis"
    return $::TCL_ERROR
}

set portList {}

chassis get      "1600T-2"
set chassis      [chassis cget -id]
set card         2
set port         1

if {![port isValidFeature $chassis $card $port $::portFeatureVcat]} {
    errorMsg " portFeatureVcat  is not supported on port $chassis $card $port "
}
else {

    port setFactoryDefaults $chassis $card $port
    port config -portMode portPosChannelizedMode
    if {[port set $chassis $card $port]} {
        errorMsg "error setting port on  $chassis $card $port"
    }

    sonet setDefault
    sonet config -interfaceType      oc192
    sonet config -useRecoveredClock  sonetNoClock
    sonet config -operation          sonetLoopback
    if {[sonet set $chassis $card $port]} {
        errorMsg "error setting sonet on  $chassis $card $port"
    }

    if {[sonetCircuitList select $chassis $card $port]} {
        errorMsg "error selecting sonetCircuitList on  $chassis $card $port"
    }

    sonetCircuitList clearAllCircuits

    sonetCircuit setDefault
    sonetCircuit config -txActiveTimeslotList "1 2"
    sonetCircuit config -rxActiveTimeslotList "1 2"
    sonetCircuit config -txIdleTimeslotList   4
    sonetCircuit config -rxIdleTimeslotList   4
    sonetCircuit config -name     "Circuit 1"
    sonetCircuit config -direction circuitBidirectionSymmetrical
    sonetCircuit config -txType    circuitPayloadRateSTS1mv
    sonetCircuit config -rxType    circuitPayloadRateSTS1mv
    sonetCircuit config -enableTxLcas false
    sonetCircuit config -enableRxLcas false
    if {[sonetCircuitList add]} {
        errorMsg "error adding circuit 1 to sonetCircuitList on  $chassis $card $port"
    }
}
```

```

set circuitId1 [sonetCircuit cget -index]

sonetCircuitProperties setDefault
sonetCircuitProperties config -payloadType sonetGfp
sonetCircuitProperties config -dataScrambling false
sonetCircuitProperties config -C2byteTransmit 22
sonetCircuitProperties config -C2byteExpected 22
sonetCircuitProperties config -rxCrc sonetCrc32
sonetCircuitProperties config -txCrc sonetCrc32
sonetCircuitProperties config -transmitMode circuitTxModePacketStreams
if {[sonetCircuitProperties set $chassis $card $port $circuitId1]} {
    errorMsg "error setting sonetCircuitProperties on $chassis $card $port for circuit
$circuitId1"
}

gfpOverhead setDefault
gfpOverhead config -enableSingleBitErrorCorrection true
gfpOverhead config -enablePayloadScrambling true
gfpOverhead config -expectedUPI 0x12
if {[gfpOverhead set $chassis $card $port $circuitId1]} {
    errorMsg "error setting gfpOverhead on $chassis $card $port for circuit $circuitId1"
}

# Configuring sonet errors for port
sonetError setDefault
sonetError config -insertionMode sonetContinuous
sonetError config -consecutiveErrors 1
sonetError config -errorPeriod 0
sonetError config -errorUnits sonetSeconds
sonetError setError sonetLofError

sonetError setDefault
sonetError config -insertionMode sonetContinuous
sonetError config -consecutiveErrors 1
sonetError config -errorPeriod 0
sonetError config -errorUnits sonetSeconds
sonetError setError sonetLineRdi
if {[sonetError set $chassis $card $port]} {
    errorMsg "error setting sonetError on $chassis $card $port"
}
# Configuring sonet errors per circuit
set timeSlot 1
sonetError setDefault
sonetError config -insertionMode sonetContinuous
sonetError config -consecutiveErrors 1
sonetError config -errorPeriod 1
sonetError config -errorUnits sonetFrames
sonetError setError sonetBip3Error
if {[sonetError set $chassis $card $port $circuitId1 $timeSlot]} {
    errorMsg "error setting sonetError on $chassis $card $port for circuit $circuitId1
for timeslot $timeSlot"
}

set timeSlot 2
sonetError config -insertionMode sonetContinuous
sonetError config -consecutiveErrors 5
sonetError config -errorPeriod 1
sonetError config -errorUnits sonetFrames
sonetError setError sonetPathRdi
if {[sonetError set $chassis $card $port $circuitId1 $timeSlot]} {

```

```

        errorMsg "error setting sonetError on $chassis $port for circuit $circuitId1
for timeslot $timeSlot"
    }

    set timeSlot 4
    sonetError setDefault
    sonetError config -insertionMode      sonetContinuous
    sonetError config -consecutiveErrors 1
    sonetError config -errorPeriod       1
    sonetError config -errorUnits        sonetFrames
    sonetError setError sonetPathLop
if {[sonetError set $chassis $port $circuitId1 $timeSlot]} {
    errorMsg "error setting sonetError on $chassis $port for circuit $circuitId1
for timeslot $timeSlot"
}
# Configuring the second circuit
    sonetCircuit setDefault
    sonetCircuit config -txActiveTimeslotList "3 9"
    sonetCircuit config -rxActiveTimeslotList 13
    sonetCircuit config -txIdleTimeslotList 10
    sonetCircuit config -rxIdleTimeslotList "22 67"
    sonetCircuit config -name "Circuit 2"
    sonetCircuit config -direction circuitBidirectionAsymmetrical
    sonetCircuit config -txType circuitPayloadRateSTS1mv
    sonetCircuit config -rxType circuitPayloadRateSTS3cmv
    sonetCircuit config -enableTxLcas true
    sonetCircuit config -enableRxLcas true
if {[sonetCircuitList add]} {
    errorMsg "error adding circuit 2 to sonetCircuitList on $chassis $card $port"
}

set circuitId2 [sonetCircuit cget -index]

sonetCircuitProperties setDefault
    sonetCircuitProperties config -payloadType      sonetFrameRelayCisco
    sonetCircuitProperties config -dataScrambling   false
    sonetCircuitProperties config -C2byteTransmit  22
    sonetCircuitProperties config -C2byteExpected  22
    sonetCircuitProperties config -rxCrc          sonetCrc16
    sonetCircuitProperties config -txCrc          sonetCrc16
    sonetCircuitProperties config -transmitMode    circuitTxModePacketStreams
if {[sonetCircuitProperties set $chassis $port $circuitId2]} {
    errorMsg "error setting sonetCircuitProperties on $chassis $card $port for circuit
$circuitId2"
}

lcas setDefault
    lcas config -rsAck      15
    lcas config -holdOff    25
    lcas config -waitToRestore 35
if {[lcas set $chassis $port $circuitId2]} {
    errorMsg "error setting lcas on $chassis $port for circuit $circuitId2"
}

# Configuring sonet errors for port
    sonetError config -insertionMode      sonetContinuous
    sonetError config -consecutiveErrors 1
    sonetError config -errorPeriod       0
    sonetError config -errorUnits        sonetSeconds
    sonetError setError sonetLineRei

```

```

sonetError setDefault
sonetError config -insertionMode      sonetContinuous
sonetError config -consecutiveErrors 1
sonetError config -errorPeriod       0
sonetError config -errorUnits        sonetSeconds
sonetError setError sonetLineRdi

if {[sonetError set $chassis $card $port]} {
    errorMsg "error setting sonetError on $chassis $card $port"
}

# Configuring sonet errors per circuit
set timeSlot 3
sonetError setDefault
sonetError config -insertionMode      sonetContinuous
sonetError config -consecutiveErrors 1
sonetError config -errorPeriod       1
sonetError config -errorUnits        sonetFrames
sonetError setError sonetPathRei

if {[sonetError set $chassis $card $port $circuitId2 $timeSlot]} {
    errorMsg "error setting sonetError on $chassis $card $port for circuit $circuitId2
for timeslot $timeSlot"
}

filter setDefault
filter config -captureTriggerCircuit filterAnyCircuit
filter config -captureFilterCircuit filterAnyCircuit
filter config -captureTriggerEnable true
filter config -captureFilterEnable true
if {[filter set $chassis $port]} {
    errorMsg "error setting filter on $chassis $port"
}

filterPallete setDefault
filterPallete config -circuitList "1 2"
if {[filterPallete set $chassis $port]} {
    errorMsg "error setting filterPallete on $chassis $port"
}

lappend portList [list $chassis $card $port]
ixWritePortsToHardware      portList
ixCheckLinkState            portList

# Circuit 1 - Stream 1
set streamId 1
stream setDefault
stream config -name      "Circuit 1"
stream config -enabletrue
gfp setDefault
gfp config -enablePli   true
gfp config -pli        65
gfp config -payloadType gfpDataFcsNullExtensionEthernet
gfp config -fcs         gfpGoodFcs
if {[gfp set $chassis $port $circuitId1]} {
    errorMsg "error setting gfp on $chassis $port for circuit $circuitId1"
}
if {[stream setCircuit $chassis $card $port $circuitId1 $streamId]} {
    errorMsg "error setting circuit stream on $chassis $card $port for circuit
$circuitId1"
}

```

```
# Circuit 2 - Stream 1
set streamId 1
stream setDefault
stream config -name      "Circuit 2"
stream config -enable    true
stream config -framesize 100
protocol setDefault
protocol config -name    ipV4
ip setDefault
ip config -precedence routine
ip config -ttl       70
ip config -ipProtocol ipV4ProtocolReserved255
if {[ip set $chassis $card $port]} {
    errorMsg "error setting ip on $chassis $card $port for circuit $circuitId2"
}

frameRelay setDefault
frameRelay config -dlci 10
if {[frameRelay set $chassis $card $port $circuitId2]} {
    errorMsg "error setting frameRelay on $chassis $card $port for circuit $circuitId2"
}
if {[streamsetCircuit $chassis $card $port $circuitId2 $streamId]} {
    errorMsg "error setting circuit stream on $chassis $card $port for circuit
$circuitId2"
}

ixWriteConfigToHardware portList -noProtocolServer
}
```

SEE ALSO

sonetCircuit, *sonetCircuitProperties*

NAME - sonetCircuitProperties

sonetCircuitProperties - used to configure circuit properties after the circuit is added. The Sonet properties for the circuit is configured here.

SYNOPSIS

sonetCircuitProperties *sub-command options*

DESCRIPTION

The **sonetCircuitProperties** command is used to configure the circuit properties of a POS port of a card on a chassis.

STANDARD OPTIONS

transitMode

Configure the transmit mode. (*default = circuitTxModePacketStreams*) Available options:

- *circuitTxModePacketStreams*
- *circuitTxModeAdvancedScheduler*

payloadType

Configure the Sonet header payload type. (*default = sonetHdlcPppIp*) Available options:

- *sonetHdlcPppIp*
- *sonetCiscoHdlc*
- *sonetOther*
- *sonetFrameRelay1490*
- *sonetFrameRelayCisco*
- *sonetGfp*
- *sonetLaps*

**dataScrambling
*true / false***

Configure the Sonet dataScrambling payload type. (*default = false*)

C2byteTransmit

Configure the Sonet C2byteTransmit . (*default = 22*)

C2byteExpected

Configure the Sonet C2byteExpected. (*default = 22*)

rxCrc

This parameter is used to configure Rx CRC.

txCrc

This parameter is used to configure Tx CRC.

index

This parameter is used to view the circuit index assigned by hardware. (*default = 0*)

COMMANDS

The **sonetCircuitProperties** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

sonetCircuitProperties **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **sonetCircuitProperties** command.

sonetCircuitProperties config option value

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS).

sonetCircuitProperties get chassisID cardID portID circuitID

Gets the IxTclHal configurations from local IxHal. Specific errors are:

- No connection to a chassis
- Invalid port number
- Not a supported feature on this port
- The port is being used by another user
- Configured parameters are not valid for this setting

sonetCircuitProperties set chassisID cardID portID circuitID

Sets the IxTclHal configurations to local IxHal.

sonetCircuitProperties setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See example on page A-667.

SEE ALSO

sonetCircuit, *sonetCircuitList*

NAME - sonetError

sonetError - configure the sonet error generation of a POS port of a card on a chassis.

SYNOPSIS

sonetError *sub-command options*

DESCRIPTION

The **sonetError** command is used to configure the sonet error generation properties of a POS port of a card on a chassis.

STANDARD OPTIONS

consecutiveErrors

The number of consecutive error frames to insert when an error is inserted either periodically, continuously or only once. (*default = 1*)

errorPeriod

If *insertionMode* is set to *sonetPeriodic*, then this is the period of time or number of frames to insert errors over, depending on the setting of *errorUnits*. A value of 1 is always used for OC12/OC3 ports. (*default = 1*)

errorUnits

If *insertionMode* is set to *sonetPeriodic*, then this determines whether *errorPeriod* refers to time (expressed in seconds) or frames. OC12/OC3 cards may only use units of seconds.

Option	Value	Notes
sonetFrames	0	(default) <i>errorPeriod</i> expressed in number of frames
sonetSeconds	1	<i>errorPeriod</i> expressed in number of seconds

insertionMode

The periodicity of error insertion.

Option	Value	Notes
sonetContinuous	0	Errors are inserted continuously
sonetPeriodic	1	Errors are inserted periodically as determined by <i>errorPeriod</i> and <i>errorUnits</i>
sonetOff	2	(default) Errors are not inserted

sonetErrorType

Read-only. When an error configuration is read back with *sonetError getError*, this reflects the sonet error type. All of the errors listed here are also non-Vcat port level errors.

Option	Value	Usage
sonetLofError	1	Loss of Frame (Vcat port level error)
sonetBip1Error	2	BIP 1 (Vcat port level error)
sonetBip2Error	3	BIP 2 (Vcat port level error)
sonetBip3Error	4	BIP 3 (circuit level error)
sonetLineAis	5	Line AIS (Vcat port level error)
sonetLineRei	6	Line REI (Vcat port level error)
sonetLineRdi	7	Line RDI (Vcat port level error)
sonetPathLop	8	Loss of Path (circuit level error)

Option	Value	Usage
sonetPathAis	9	Path AIS (circuit level error)
sonetPathRei	10	Path REI (circuit level error)
sonetPathRdi	11	Path RDI (circuit level error)
sonetLosError	12	Loss of signal

COMMANDS

The **sonet** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

sonetError cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **sonet** command.

sonetError config option value

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

sonetError get chasID cardID portID circuitID timeslot

Gets the current *sonetError* configuration for all of the sonet error types for the port indicated from its hardware. Call this command before calling **sonet cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is not a Packet over Sonet port.

sonetError getError sonetErrorType

Retrieves the values of the attributes (*insertionMode*, *consecutiveErrors*, *errorPeriod*, and *errorUnits*) associated with the *sonetErrorType*. See the description of *sonetErrorType* above for a list of the possible values. The *sonetError get* command must be used before *getError*.

sonetError insertError sonetErrorType chasID cardID portID circuitID timeslot

Inserts a single instance of the error defined by *sonetErrorType* and in the standard options into the sonet stream for the indicated port.

sonetError set chasID cardID portID circuitID timeslot

Sets the configuration of the port in IxTclHAL with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **sonet config option value** command. Specific per-port errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- The port is not a Packet over Sonet port or 10Gigabit WAN.

These error types can be per-circuit:

- sonetBip3Error
- sonetPathLop

- sonetPathAis
- sonetPathRei
- sonetPathRdi

sonetError setDefault

Sets to IxTclHal default values for all configuration options.

sonetError setError *sonetErrorType*

Sets the attributes (*insertionMode*, *consecutiveErrors*, *errorPeriod*, and *errorUnits*) associated with the *sonetErrorType*. See the description of *sonetErrorType* above for a list of the possible values. The *sonetError set* command should be used after this command to write the values to the hardware.

sonetError start *chassisID cardID portID circuitID*

Starts sonet error insertion on the selected port. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The port is not a Packet over Sonet port or 10Gigabit WAN.

sonetError stop *chassisID cardID portID circuitID*

Stops sonet error insertion on the selected port. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The port is not a Packet over Sonet port or 10Gigabit WAN.

EXAMPLES

```
package require IxTclHal

proc printState {} \
{
    for {set errType $::sonetLoSError} {$errType <= $::sonetPathRei} {
        incr errType
        sonetError getError $errType
        ixPuts -nonewline " errType: $errType"
        ixPuts -nonewline " insertionMode: "
        ixPuts -nonewline [sonetError cget -insertionMode]
        ixPuts -nonewline " errorPeriod: "
        ixPuts -nonewline [sonetError cget -errorPeriod]
        ixPuts -nonewline " errorUnits: "
        ixPuts -nonewline [sonetError cget -errorUnits]
        ixPuts -nonewline " consecutiveErrors: "
        ixPuts [sonetError cget -consecutiveErrors]
    }
}

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
```

```

        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that an OC48 POS card is in slot 17
set card 17
set portList [list [list $chas $card 1]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the type of card and check if it's the correct type
set ifType [card getInterface $chas $card]
if {$ifType != $::interfaceOC48} {
    ixPuts "Card $card is not an OC48c POS card"
} else {
    sonetError setDefault
    ixPuts ""
    ixPuts "Initial State:"
    printState

    sonetError config -insertionMode sonetContinuous
    sonetError config -consecutiveErrors 5
    sonetError setError sonetLofError

    sonetError config -insertionMode sonetPeriodic
    sonetError config -errorUnits sonetSeconds
    sonetError config -errorPeriod 10
    sonetError config -consecutiveErrors 20
    sonetError setError sonetBip1Error

    sonetError set $chas $card 1
    ixWriteConfigToHardware portList
    sonetError get $chas $card 1
    ixPuts "After changes:"
    printState

    sonetError start $chas $card 1
    after 1000
    sonetError stop $chas $card 1

    sonetError setDefault
    sonetError config -consecutiveErrors 4
    sonetError insertError sonetLineAis $chas $card 1
}
# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server

```

```
if [isUNIX] {  
    ixDisconnectTclServer $host  
}
```

SEE ALSO

card, port, sonet

NAME - sonetOverhead

sonetOverhead - insert J0/J1 overhead trace messages.

SYNOPSIS	sonetOverhead <i>sub-command options</i>
DESCRIPTION	The sonetOverhead command is used to configure the insertion of trace messages in the J0/J1 areas of the sonet overhead. The sonetOverhead configurations for circuit are grouped under the sonetCircuitProperties command.
<hr/>	
STANDARD OPTIONS	
enableJ0Insertion <i>true / false</i>	If <i>true</i> , the message in <i>traceMessageJ0</i> is inserted in the sonet header. (<i>default = false</i>)
enableJ1Insertion <i>true / false</i>	If <i>true</i> , the message in <i>traceMessageJ1</i> is inserted in the sonet header. (<i>default = false</i>)
traceMessageJ0	The value of the trace message to insert in the J0 bytes of the sonet header, if <i>enableJ0Insertion</i> is set to <i>true</i> . The value is expressed as a hex string. After a <i>sonetOverhead get</i> , this holds the value of the J0 bytes from the received sonet header. (<i>default = {}</i>)
traceMessageJ1	The value of the trace message to insert in the J1 bytes of the sonet header, if <i>enableJ1Insertion</i> is set to <i>true</i> . The value is expressed as a hex string. After a <i>sonetOverhead get</i> , this holds the value of the J1 bytes from the received sonet header. (<i>default = {}</i>)
<hr/>	
COMMANDS	The sonet command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
<hr/>	
sonetOverhead cget <i>option</i>	
Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the sonet command.	
sonetOverhead config <i>option value</i>	
Modify the configuration options of the port. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.	
sonetOverhead get <i>chassisID cardID portID [circuitID] [timeslot]</i>	
Gets the current <i>sonetOverhead</i> trace messages for the port indicated from its hardware. Call this command before calling sonet cget <i>option value</i> to get the value of the configuration option. Specific errors are:	
<ul style="list-style-type: none"> • No connection to a chassis • Invalid port number • The port is not a Packet over Sonet port. 	

sonetOverhead set *chassisID cardID portID [circuitID] [timeslot]*

Sets the configuration of the indicated port by reading the configuration option values set by the **sonet config *option value*** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- The port is not a Packet over Sonet port or 10Gigabit WAN.

sonetOverhead setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```

package req IxTclHal
set hostname astro
set txCard 2
set rxCard 3

set port 1
set streamId 1

set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chassis [ixGetChassisID $host]

set portList [list [list $chassis $txCard $port] [list $chassis
$rxCard $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# port setup

# sonet setup

sonetOverhead setDefault
sonetOverhead config -enableJ0Insertion true

```

```
sonetOverhead config -enableJ1Insertion true
sonetOverhead config -traceMessageJ0 {F8 00 00 00 00 00 00
00 00 00 00 00 00 00}
sonetOverhead config -traceMessageJ1 {E8 65 6C 6C 6F 20 74
68 65 72 65 20 00 00 00}
sonetOverhead set $chassis $txCard $port
ixWritePortsToHardware portList

sonetOverhead get $chassis $rxCard
$port

set j0 [sonetOverhead cget -traceMessageJ0]
set j1 [sonetOverhead cget -traceMessageJ1]

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

card, port, sonet, sonetCircuitProperties

NAME - splitPacketGroup

splitPacketGroup - configures split packet group operation.

SYNOPSIS

splitPacketGroup *sub-command options*

DESCRIPTION

The **splitPacketGroup** command is used to configure split packet groups in stream generation. Up to 17 bytes can be configured in three separate chunks.

Note: When configuring split packet groups, and all 17 bytes are being used, it is necessary to reset the defaults for the split packet group before changing the size of each split section.

STANDARD OPTIONS

groupIdOffset The offset, in bytes, from the starting point set in *groupIdOffsetBaseType*.

groupIdOffset BaseType Where in the packet to start the offset for the PGID split section.

Option	Value	Usage
<i>splitPgidStartOfFrame</i>	0	Start offset at the beginning of the frame
<i>splitPgidOffsetFrom Signature</i>	1	Start offset from the beginning of the signature.

groupIdWidth The number of bytes in the PGID split section. (*default = 4*)

groupIdMask The bit mask for the PGID group. (*default = FF FF FF FF*)

COMMANDS

The **splitPacketGroup** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

splitPacketGroup **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **splitPacketGroup** command.

splitPacketGroup **config** *option value*

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

splitPacketGroup **get** *chassisID cardID portID*

Gets the current split PGID configuration for the port indicated from its hardware. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is not a Packet over Sonet port.

splitPacketGroup set *chassisID cardID portID*

Sets the configuration of the indicated port by reading the configuration option values set by the config *option value* command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port

splitPacketGroup setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package req IxTclHal
proc tdisp {val} {
    if {$val >= 1} {
        return "$val seconds"
    }

    if {$val < 1 && $val >= 0.001} {
        return [format "%0.4f ms" [expr 1e003 * $val]]
    }

    if {$val < 0.001} {
        return [format "%0.2f us" [expr 1e006 * $val]]
    }
}

proc testDuration {val} {
    set decimal [string range [format "%0.4f" [expr $val - /
int($val)]] 2 end]

    return "[clock format [expr int($val)] -format /
"%H:%M:%S" -gmt 1]"

    return "[clock format [expr int($val)] -format /
"%H:%M:%S" -gmt 1].$decimal"
}

proc scaleChange {numbits} {
    global maxDurationWidget
    global ticksPerSample samplePeriod sampleDuration /
testDuration splitPgIdVals

    set ticksPerSample [expr int(pow(2,$numbits))]
    set samplePeriod [expr (20.0 * $ticksPerSample) /
1000000000]

    set sampleDuration [expr $samplePeriod * 0x20000]
    set testDuration [testDuration $sampleDuration]
    set samplePeriod [tdisp $samplePeriod]
    set val [mpexpr 131071 << $numbits]
    set mask [mpformat %012x [mpexpr (0xffffffffffff ^ $val)]]
    regsub "0x" $mask ""
    set wordHi "[string range $mask 0 1] /
[string range $mask 2 3] [string range $mask 4 5] [string range
$mask 6 7]"
```

```

        set wordLo  "[string range $mask 8 9] [string range /
$mask 10 11]"

        set splitPgidVals "$wordHi $wordLo"
        puts "ticksPerSample      $ticksPerSample   "
        puts "samplePeriod        $samplePeriod     "
        puts "sampleDuration       $sampleDuration   "
        puts "testDuration         $testDuration     "
        puts "samplePeriod         $samplePeriod     "
        puts "splitPgidVals       $splitPgidVals"
    }

proc setSplitPacketGroup {port offset} {

    global splitPgidVals
    scan $port "%d %d %d" ch ca po
    packetGroup getRx $ch $ca $po
    packetGroup config -groupIdMode          packetGroupSplit
    packetGroup setRx $ch $ca $po
    splitPacketGroup setDefault
    splitPacketGroup set $ch $ca $po 0
    splitPacketGroup set $ch $ca $po 1
    splitPacketGroup set $ch $ca $po 2
    splitPacketGroup config -groupIdOffset    [expr $offset + 4]
    splitPacketGroup config -groupIdMask      /
[lrange $splitPgidVals 4 5]

    splitPacketGroup config -groupIdWidth      2
    splitPacketGroup set $ch $ca $po 0
    splitPacketGroup setDefault
    splitPacketGroup config -groupIdOffset      $offset
    splitPacketGroup config -groupIdWidth      4
    splitPacketGroup config -groupIdMask      /
[lrange $splitPgidVals 0 3]

    splitPacketGroup set $ch $ca $po 1
    splitPacketGroup setDefault
    splitPacketGroup set $ch $ca $po 2
}

```

SEE ALSO*packetGroup*

NAME - srpArp

srpArp - configure an SRP ARP packet

SYNOPSIS

srpArp *sub-command options*

DESCRIPTION

The **srpArp** command is used to configure the contents of an SRP ARP packet to be transmitted as part of a stream.

STANDARD OPTIONS

mode

Indicates the mode of the packet.

Option	Value	Usage
<i>srpModeReserved000</i>	0	
<i>srpModeReserved001</i>	1	
<i>srpModeReserved010</i>	2	
<i>srpModeATMCell</i>	3	An ATM data cell.
<i>srpModeControlMessage1</i>	4	A control message to be passed to the destination host.
<i>srpModeControlMessage2</i>	5	A control message to be buffered for the destination host.
<i>srpModeUsageMessage</i>	6	(default) An SRP usage message.
<i>srpModePacketData</i>	7	An SRP data packet.

parityBit

The parity over the other SRP header bits.

Option	Value	Usage
<i>srpParityBitEven</i>	0	Insert an even parity bit.
<i>srpParityBitOdd</i>	1	(default) Insert a correct, odd parity.

priority

Indicates the priority of the SRP packet. Eight priority levels (0 through 7) are offered. Packets on the ring are treated as low or high priority, where a threshold variable determines which values fall into the high priority range. This value is usually copied from the IP precedence bits. Control packets always use priority 7. (default = 0)

ringIdentifier

Indicates whether the inner or outer ring is to receive the packet. Used by the Arp to make decisions about ring wrap or to determine whether or not a packet is accepted on the ring.

Option	Value	Usage
<i>srpRingIdentifierOuter</i>	0	(default) Outer ring.
<i>srpRingIdentifierInner</i>	1	Inner ring.

ttl

The hop counter decremented each time a node forwards a packet. When the counter reaches 0, the packet is removed from the ring. This may be set from 0 through 255. (default = 1)

COMMANDS

The **srpArp** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

srpArp cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **srpArp** command.

srpArp config option value

Modify the configuration options of the srpArp. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for srpArp.

srpArp decode capSlice chasID cardID portID

Decodes a captured slice/frame into the srpArp variables. If not an **srpArp** frame, returns TCL_ERROR. May be used to determine if the captured frame is a valid srpArp frame. Specific errors are:

- No connection to a chassis
- The captured frame is not an srpArp frame

srpArp get chasID cardID portID

Gets the current configuration of the srpArp frame for port with id *portID* on card *cardID*, chassis *chasID*. from its hardware. Call this command before calling **srpArp cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

srpArp set chasID cardID portID

Sets the configuration of the srpArp in IxHAL for port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **srpArp config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

srpArp setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
```

```

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 71
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

sonet get $chas $card $port
sonet config -header          sonetSrp
sonet set $chas $card $port
sonet write $chas $card $port

#
# IPS
srpIps setDefault

# Set Srp Header
srpIps config -ttl           128
srpIps config -priority      2
srpIps config -mode
srpIps config -ringIdentifier
srpIps config -parityBit     srpParityBitOdd

# Set Control Header
srpIps config -controlVersion 0
srpIps config -controlTTL    128

# Set IPS Specific Parameters
srpIps config -originatorMacAddress      [stream cget -sa]
srpIps config -requestType
srpIpsRequestTypeWaitToRestore
srpIps config -pathIndicator
srpIpsPathIndicatorShort
srpIps config -statusCode             srpIpsStatusCodeIdle

protocol setDefault
protocol config -appName SrpIps

srpIps set $chas $card $port
stream set $chas $card $port 1

#
# Discovery
#
srpDiscovery setDefault

# Set Srp Header

```

```

srpDiscovery config -ttl          128
srpDiscovery config -priority      2
srpDiscovery config -mode          srpModeControlMessage1
srpDiscovery config -ringIdentifier    srpRingIdentifierInner
srpDiscovery config -parityBit      srpParityBitOdd

# Set Control Header
srpDiscovery config -controlVersion     0
srpDiscovery config -controlTTL        128

# Set Discovery Specific Parameters
srpDiscovery config -originatorMacAddress [stream cget -sa]
srpDiscovery config -topologyLength      25

# Set MAC bindings
srpDiscovery clearAllMacBindings
srpMacBinding config -address           {00 00 de b0 01 00}
srpMacBinding config -wrappedNode      srpWrappedNode
srpMacBinding config -ringIdentifier    srpRingIdentifierInner
srpDiscovery addMacBinding

srpMacBinding config -address           {00 00 de b0 01 01}
srpMacBinding config -wrappedNode      srpWrappedNode
srpMacBinding config -ringIdentifier    srpRingIdentifierInner
srpDiscovery addMacBinding

srpDiscovery set $chas $card $port

protocol setDefault
protocol config -appName SrpDiscovery

stream set $chas $card $port 2

#
# ARP
#
srpArp setDefault

# Set Srp Header
srpArp config -ttl          255
srpArp config -priority      7
srpArp config -mode          srpModePacketData
srpArp config -ringIdentifier    srpRingIdentifierOuter
srpArp config -parityBit      srpParityBitEven

srpArp set $chas $card $port

protocol setDefault
protocol config -appName srpArp

stream set $chas $card $port 3

#
# Usage
#
port get $chas $card $port
srpUsage setDefault

# Set Srp Header
srpUsage setDefault
srpUsage config -ttl          128
srpUsage config -priority      2
srpUsage config -mode          srpModeUsageMessage

```

```
srpUsage config -ringIdentifier           srpRingIdentifierInner
srpUsage config -parityBit               srpParityBitOdd

srpUsage config -txMacAddress           [port cget -MacAddress]
srpUsage config -txUsageEnable          true
srpUsage config -txRepeatInterval      100
srpUsage config -txValue                0
srpUsage config -rxMacAddress           [port cget -
DestMacAddress]
srpUsage config -rxTimeout              100
srpUsage config -rxTimeoutThreshold    10
srpUsage set $chassis $card $port

port set $chassis $card $port
port write $chassis $card $port

# Post transmission.
stat get statAllStats $chassis $card $port
stat cget -srpKeepAliveFramesReceived
stat getRate $chassis $card $port -srpSrpHeaderParityErrors

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

srpDiscovery, srpMacBinding, srpIps, srpUsage

NAME - srpDiscovery

srpDiscovery - configure an SRP discovery packet

SYNOPSIS

srpDiscovery *sub-command options*

DESCRIPTION

The **srpDiscovery** command is used to configure the contents of an SRP discovery packet to be transmitted as part of a stream. The bindings are configured in the *srpMacBinding* command and then added to the discovery packet using the *addMacBinding* sub-command.

STANDARD OPTIONS

controlCheckSumMode

The checksum mode associated with the control packet.

Option	Value	Usage
<i>srpDiscoveryCheckSumBad</i>	0	Insert a bad checksum.
<i>srpDiscoveryCheckSumGood</i>	1	(default) Insert a good checksum.

controlTTL

The control layer hop-count that is decremented by one each time a node forwards a control packet. (*default = 0*)

controlType

An alternate setting for the control type setting in the packet; *controlTypeOverride* must be set to *true* for this value to be used. (*default = 1*)

controlTypeOverride *true / false*

Indicates whether the value in *controlType* should be used to override the default setting of *srpControlTypeDiscovery*. (*default = false*)

controlVersion

The version number associated with the control type fields. The only supported version is version 0. (*default = 0*)

controlVersionOverride *true / false*

Indicates whether the value in *controlVersion* should be used to override the default setting of 0. (*default = false*)

mode

Indicates the mode of the packet.

Option	Value	Usage
<i>srpModeReserved000</i>	0	
<i>srpModeReserved001</i>	1	
<i>srpModeReserved010</i>	2	
<i>srpModeATMCell</i>	3	An ATM data cell.
<i>srpModeControlMessage1</i>	4	A control message to be passed to the destination host.
<i>srpModeControlMessage2</i>	5	A control message to be buffered for the destination host.
<i>srpModeUsageMessage</i>	6	(default) An SRP usage message.
<i>srpModePacketData</i>	7	An SRP data packet.

originatorMacAddress

The original source MAC address. This differs from the source MAC address in that as a packet is forwarded from node to node, the source MAC address is modified to reflect the current node, whereas the originator MAC address always reflects the first source address. (*default* = {00 00 00 00 00 00})

parityBit

The parity over the other SRP header bits.

Option	Value	Usage
<i>srpParityBitEven</i>	0	Insert an even parity bit.
<i>srpParityBitOdd</i>	1	(<i>default</i>) Insert a correct, odd parity.

priority

Indicates the priority of the SRP packet. Eight priority levels (0 through 7) are offered. Packets on the ring are treated as low or high priority, where a threshold variable determines which values fall into the high priority range. This value is usually copied from the IP precedence bits. Control packets always use priority 7. (*default* = 0)

ringIdentifier

Indicates whether the inner or outer ring is to receive the packet. Used by the Discovery to make decisions about ring wrap or to determine whether or not a packet is accepted on the ring.

Option	Value	Usage
<i>srpRingIdentifierOuter</i>	0	(<i>default</i>) Outer ring.
<i>srpRingIdentifierInner</i>	1	Inner ring.

topologyLength

The length of the topology discovery packet beginning with the MAC type/MAC binding data. This must be a multiple of seven since each binding is seven bytes long. (*default* = 0)

ttl

The hop counter decremented each time a node forwards a packet. When the counter reaches 0, the packet is removed from the ring. This may be set from 0 through 255. (*default* = 1)

COMMANDS

The *srpDiscovery* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

srpDiscovery addMacBinding

Adds the MAC binding found in the *srpMacBinding* command to the list associated with the discovery packet.

srpDiscovery cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *srpDiscovery* command.

srpDiscovery clearAllMacBindings

Clears all of the MAC bindings associated with the discovery packet.

srpDiscovery config option value

Modify the configuration options of the *srpDiscovery*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *srpDiscovery*.

srpDiscovery decode *capSlice chassisID cardID portID*

Decodes a captured slice/frame into the srpDiscovery variables. If not an *srpDiscovery* frame, returns TCL_ERROR. May be used to determine if the captured frame is a valid srpDiscovery frame. Specific errors are:

- No connection to a chassis
- The captured frame is not an srpDiscovery frame

srpDiscovery delMacBinding *macBindingAddress*

Deletes the MAC binding which matches *macBindingAddress*.

srpDiscovery get *chassisID cardID portID*

Gets the current configuration of the srpDiscovery frame for port with id *portID* on card *cardID*, chassis *chassisID*. from its hardware. Call this command before calling *srpDiscovery cget option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

srpDiscovery getFirstMacBinding

Accesses the first MAC binding in the list and moves the data to the *srpMacBinding* command.

srpDiscovery getMacBinding *macBindingAddress*

Accesses the MAC binding in the list which uses *macBindingAddress* and moves the data to the *srpMacBinding* command.

srpDiscovery getNextMacBinding

Accesses the next MAC binding in the list and moves the data to the *srpMacBinding* command.

srpDiscovery set *chassisID cardID portID*

Sets the configuration of the srpDiscovery in IxHAL for port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the *srpDiscovery config option value* command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

srpDiscovery setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *srpArp*.

SEE ALSO

srpMacBinding, srpArp, srpIps, srpUsage

NAME - srpIps

srpIps - configure an SRP IPS protection control packet

SYNOPSIS	<i>srpIps sub-command options</i>
-----------------	-----------------------------------

DESCRIPTION	The srpIps command is used to configure the contents of an SRP Intelligent Protection Switching (IPS) packet to be transmitted as part of a stream.
--------------------	--

STANDARD OPTIONS

controlCheckSumMode The checksum mode associated with the control packet.

Option	Value	Usage
<i>srplpsCheckSumBad</i>	0	Insert a bad checksum.
<i>srplpsCheckSumGood</i>	1	(default) Insert a good checksum.

controlTTL The control layer hop-count that is decremented by one each time a node forwards a control packet. (*default = 0*)

controlType An alternate setting for the control type setting in the packet; *controlTypeOverride* must be set to *true* for this value to be used. (*default = 2*)

**controlTypeOverride
*true / false*** Indicates whether the value in *controlType* should be used to override the default setting of *srpControlTypeIps*. (*default = false*)

controlVersion The version number associated with the control type fields. The only supported version is version 0. (*default = 0*)

**controlVersionOverride
*true / false*** Indicates whether the value in *controlVersion* should be used to override the default setting of 0. (*default = false*)

mode Indicates the mode of the packet.

Option	Value	Usage
<i>srpModeReserved000</i>	0	
<i>srpModeReserved001</i>	1	
<i>srpModeReserved010</i>	2	
<i>srpModeATMCell</i>	3	An ATM data cell.
<i>srpModeControlMessage1</i>	4	A control message to be passed to the destination host.
<i>srpModeControlMessage2</i>	5	A control message to be buffered for the destination host.
<i>srpModeUsageMessage</i>	6	(default) An SRP usage message.
<i>srpModePacketData</i>	7	An SRP data packet.

originatorMacAddress The original source MAC address. This differs from the source MAC address in that as a packet is forwarded from node to node, the source MAC address is

modified to reflect the current node, whereas the originator MAC address always reflects the first source address. (*default = {00 00 00 00 00 00}*)

parityBit

The parity over the other SRP header bits.

Option	Value	Usage
<i>srpParityBitEven</i>	0	Insert an even parity bit.
<i>srpParityBitOdd</i>	1	(<i>default</i>) Insert a correct, odd parity.

pathIndicator

Determines whether the control packet is sent only to an adjacent node or around the entire ring.

Option	Value	Usage
<i>srplpsPathIndicatorShort</i>	0	Message is just send to the next node.
<i>srplpsPathIndicatorLong</i>	1	(<i>default</i>) Message is sent around the entire ring.

priority

Indicates the priority of the SRP packet. Eight priority levels (0 through 7) are offered. Packets on the ring are treated as low or high priority, where a threshold variable determines which values fall into the high priority range. This value is usually copied from the IP precedence bits. Control packets always use priority 7. (*default = 0*)

requestType

The type of IPS request.

Option	Value	Usage
<i>srplpsRequestTypeNoRequest</i>	0	(<i>default</i>) No request.
<i>srplpsRequestTypeWaitToRestore</i>	5	Wait to restore. Instead of unwrapping immediately after a failure condition or manual request has been cleared, the node waits for a configured period of time before unwrapping
<i>srplpsRequestTypeManualSwitch</i>	6	Manual switch. Force a switch.
<i>srplpsRequestTypeSignalDegrade</i>	8	Signal degrade. A switch initiated by detecting line BER above a specific threshold or excessive CRC errors.
<i>srplpsRequestTypeSignalFail</i>	11	Signal fail. A switch initiated by detecting los of signal, los of frame, line bit error rate above a specific threshold, line AIS or excessive CRC errors.
<i>srplpsRequestTypeForcedSwitch</i>	13	Forced switch. Force a switch; same as manual switch but with higher priority.

ringIdentifier

Indicates whether the inner or outer ring is to receive the packet. Used by the IPS to make decisions about ring wrap or to determine whether or not a packet is accepted on the ring.

Option	Value	Usage
<i>srpRInIdentifierOuter</i>	0	(<i>default</i>) Outer ring.
<i>srpRInIdentifierInner</i>	1	Inner ring.

statusCode

Indicates the state of a node in terms of traffic wrapping.

Option	Value	Usage
<i>srplpsStatusCodeIdle</i>	0	(<i>default</i>) The node is prepared to perform protection switching if necessary.

Option	Value	Usage
<i>srplpsStatusCodeProtection</i>	2	Indicates that a node is currently participating in a protection switching operation.

ttl

The hop counter decremented each time a node forwards a packet. When the counter reaches 0, the packet is removed from the ring. This may be set from 0 through 255. (*default = 1*)

COMMANDS

The *srpIps* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

srpIps cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *srpIps* command.

srpIps config option value

Modify the configuration options of the srpIps. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for srpIps.

srpIps decode capSlice [chassisID cardID portID]

Decodes a captured slice/frame into the srpIps variables. If not an *srpIps* frame, returns TCL_ERROR. May be used to determine if the captured frame is a valid srpIps frame. Specific errors are:

- No connection to a chassis
- The captured frame is not an srpIps frame

srpIps get chassisID cardID portID

Gets the current configuration of the srpIps frame for port with id *portID* on card *cardID*, chassis *chassisID*, from its hardware. Call this command before calling *srpIps cget option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

srpIps set chassisID cardID portID

Sets the configuration of the srpIps in IxHAL for port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the *srpIps config option value* command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting

srpIps setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *srpArp*.

SEE ALSO

srpArp, *srpDiscovery*, *srpMacBinding*, *srpUsage*

NAME - srpMacBinding

srpMacBinding - configure an SRP Mac Binding

SYNOPSIS srpMacBinding *sub-command options*

DESCRIPTION The **srpMacBinding** command is used to configure a MAC binding that appears in an SRP Discovery packet. These bindings are included in a discovery packets through the use of the *srpDiscovery* command.

STANDARD OPTIONS

address The MAC addressed bound. (*default = {CA CA CA CA CA CA}*)

ringIdentifier Which ring the binding applies to.

Option	Value	Usage
<i>srpRingIdentifierInner</i>	0	(<i>default</i>) The inner ring.
<i>srpRingIdentifierInner</i>	1	The outer ring.

wrappedNode Whether the node is wrapped or not.

Option	Value	Usage
<i>srpUnwrappedNode</i>	0	(<i>default</i>) The node is not wrapped.
<i>srpWrappedNode</i>	1	The node is wrapped.

COMMANDS

The **srpMacBinding** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

srpMacBinding setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *srpArp*.

SEE ALSO

srpDiscovery, srpArp, srpIps, srpUsage

NAME - srpUsage

srpUsage - configure an SRP Usage packet

SYNOPSIS	srpUsage <i>sub-command options</i>
-----------------	-------------------------------------

DESCRIPTION	The srpUsage command is used to configure the contents of an SRP Usage packet to be transmitted at intervals.
--------------------	--

STANDARD OPTIONS

mode Indicates the mode of the packet.

Option	Value	Usage
<i>srpModeReserved000</i>	0	
<i>srpModeReserved001</i>	1	
<i>srpModeReserved010</i>	2	
<i>srpModeATMCell</i>	3	An ATM data cell.
<i>srpModeControlMessage1</i>	4	A control message to be passed to the destination host.
<i>srpModeControlMessage2</i>	5	A control message to be buffered for the destination host.
<i>srpModeUsageMessage</i>	6	(default) An SRP usage message.
<i>srpModePacketData</i>	7	An SRP data packet.

parityBit The parity over the other SRP header bits.

Option	Value	Usage
<i>srpParityBitEven</i>	0	Insert an even parity bit.
<i>srpParityBitOdd</i>	1	(default) Insert a correct, odd parity.

priority Indicates the priority of the SRP packet. Eight priority levels (0 through 7) are offered. Packets on the ring are treated as low or high priority, where a threshold variable determines which values fall into the high priority range. This value is usually copied from the IP precedence bits. Control packets always use priority 7. (default = 0)

ringIdentifier Indicates whether the inner or outer ring is to receive the packet. Used by the Usage to make decisions about ring wrap or to determine whether or not a packet is accepted on the ring.

Option	Value	Usage
<i>srpRingIdentifierOuter</i>	0	(default) Outer ring.
<i>srpRingIdentifierInner</i>	1	Inner ring.

rxMacAddress The source MAC address for the usage packet. (default = {00 00 00 00 00 00})

rxTimeout	The time interval, in microseconds, between SRP usage packets, which serve a keep-alive function. This must be between 8 to 65000 and should be set to approximately 106 microseconds. (<i>default = 106</i>)
rxTimeoutThreshold	The number of timeout values that can pass before the sending interface is considered down. This value must be between 1 and 31 and should be set to 16. (<i>default = 16</i>)
ttl	The hop counter decremented each time a node forwards a packet. When the counter reaches 0, the packet is removed from the ring. This may be set from 0 through 255. (<i>default = 1</i>)
txMacAddress	The MAC address of the destination to which usage packets are sent. (<i>default = {00 00 00 00 00 00}</i>)
txRepeatInterval	The interval within which usage frames are sent to upstream nodes; expressed in microseconds. This value must be between 10 and 65000. (<i>default = 106</i>)
txReserved	A reserved field, that should be set to 0. (<i>default = 0</i>)
txUsageEnabled <i>true / false</i>	If set to <i>true</i> , periodic SRP usage packets are transmitted. (<i>default = false</i>)
txValue	The value associated with the usage packet, between 0 and 65535. (<i>default = 65535</i>)

COMMANDS

The **srpUsage** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

srpUsage cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **srpUsage** command.

srpUsage config option value

Modify the configuration options of the srpUsage. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for srpUsage.

srpUsage get chassisID cardID portID

Gets the current configuration of the srpUsage frame for port with id *portID* on card *cardID*, chassis *chassisID*. from its hardware. Call this command before calling **srpUsage cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number

srpUsage set chassisID cardID portID

Sets the configuration of the srpUsage in IxHAL for port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **srpUsage config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user

A***srpUsage***

- Configured parameters are not valid for this setting

srpUsage setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *srpArp*.

SEE ALSO

srpArp, *srpDiscovery*, *srpIps*

NAME - stackedVlan

stackedVlan - configure a stack of VLAN entries

SYNOPSIS	stackedVlan <i>sub-command options</i>
DESCRIPTION	<p>The stackedVlan command is used to configure an ordered stack of VLAN entries. This command is only used when the <i>enable802dot1qTag</i> in the <i>protocol</i> command is set to <i>vlanStacked</i>. Elements of the stack are constructed in the <i>vlan</i> command. The top two elements of the stack are always present and may be modified by using the <i>setVlan</i> sub-command. Other elements are added to the bottom of the stack using <i>addVlan</i>; they may later be modified with the <i>setVlan</i> sub-command.</p> <p>The top two VLANs in a stack may be configured to increment or decrement their VLAN ID. They may either increment/decrement independently or operate in a special nested mode. To use nested mode, the top (outer) VLAN should be set to one of the non-nested increment/decrements modes and the second (inner) VLAN should be set to the nested increment or decrement mode. In this mode the inner VLAN's ID changes most rapidly.</p>
<hr/>	
STANDARD OPTIONS	
<p>numVlans <i>Read-only</i>. The number of VLANs in the stack.</p>	
<hr/>	
COMMANDS	<p>The stackedVlan command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.</p>
<p>stackedVlan addVlan</p> <p>The VLAN specification found in the <i>vlan</i> command is pushed onto the bottom of the stack. Any use of increment/decrement modes is ignored. Specific errors are:</p> <ul style="list-style-type: none"> • The VLAN could not be added. 	
<p>stackedVlan cget <i>option</i></p> <p>Returns the current value of the configuration option given by <i>option</i>. <i>Option</i> may have any of the values accepted by the stackedVlan command.</p>	
<p>stackedVlan decode <i>capSlice chasID cardID portID</i></p> <p>Decodes a captured slice/frame that contains VLAN(s), populating the <i>vlan</i> command with the top VLAN in the stack. Other VLANs may be accessed by using the <i>getFirstVlan</i>, <i>getNextVlan</i> and <i>getVlan</i> sub-commands. Specific errors are:</p> <ul style="list-style-type: none"> • No connection to a chassis • The captured frame does not contain any VLANs. 	
<p>stackedVlan delVlan <i>index</i></p> <p>Deletes the VLAN from the stack at the <i>index</i>'d position. The top of the stack is numbered 1. The top two stack elements may not be deleted. Specific errors include:</p>	

- The top two stack elements may not be deleted.
- There is no VLAN at the *index*'d position.

stackedVlan **get** *chasID cardID portID*

Gets the current configuration of the stackedVlan frame for port with id *portID* on card *cardID*, chassis *chasID*. from its hardware. Call this command before calling **stackedVlan cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is not available.
- Stacked VLANs are not supported by the port.
- Stacked VLAN data is not available; a *stream* get may be needed.

stackedVlan **getFirstVlan**

Retrieves the first VLAN from the stack; the values are available in the *vlan* command. Specific errors include:

- There are no VLANs in the list.

stackedVlan **getNextVlan**

Retrieves the next VLAN from the stack; the values are available in the *vlan* command. Specific errors include:

- There are no more VLANs in the list.

stackedVlan **getVlan index**

Retrieves the VLAN from the stack at the *index*'d position; the values are available in the *vlan* command. The top of the stack is numbered 1. Specific errors include:

- There is no VLAN at the *index*'d position.

stackedVlan **set** *chasID cardID portID*

Sets the configuration of the stackedVlan in IxHAL for port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **stackedVlan config option value** command. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- Configured parameters are not valid for this setting
- Stacked VLANs are not supported by this port.

stackedVlan **setDefault**

Sets to IxTclHal default values for all configuration options.

stackedVlan **setVlan index**

Sets the VLAN from the stack at the *index*'d position using the values from the *vlan* command. The top of the stack is numbered 1. This may be used to change an existing VLAN stack element in place. Specific errors include:

- There is no VLAN at the *index*'d position.

EXAMPLES

See examples under *vlan*.

SEE ALSO

protocol, vlan

NAME - stat

stat - gets the statistics on a port of a card on a chassis.

SYNOPSIS

stat *sub-command options*

DESCRIPTION

The **stat** command is used to get statistics. Statistics may be gathered in several ways. All statistics may be obtained through the use of the *stat get statAllStats <chassis> <card> <port>* followed by calls to get the data using *stat cget -statName*. All rate statistics may be obtained through the use of the *stat getRate statAllStats <chassis> <card> <port>* followed by calls to get the data using *stat cget -name*.

An individual statistic may be collected through the use of the *stat get statName <chassis> <card> <port>* followed by *stat cget -statName*. Note that the *statName* is formed from the standard option name by prepending *stat* to the name and capitalizing the first letter of the option. (For example, for the option **framesSent**, the *statName* is **statFramesSent**.)

Values are available through the STANDARD OPTIONS following the *stat cget* call. When using *stat cget -statName*, only those statistics valid for that type of port are returned; all others return an error (see the *enableValidStats* option). Refer to the *Ixia Reference Guide* for a list of which statistics are available for particular card modules and under particular circumstances.

For channelized BERT cards, it is necessary to request statistics for a specific channel at a time using the *getBertChannel* and *getBertChannelRate* sub-commands. These commands take a *level* description, which is discussed in [bert](#) and [bertErrorGeneration](#) on page 5-27.

STANDARD OPTIONS

Standard Options controlling statistics modes and operation

enableArpStats
true/false

Enables the collection of Arp statistics. (*default = true*) The following statistics are controlled by this option:

TxArpRequest	TxArpReply
RxArpRequest	RxArpReply

enableAtmOamStats
true/false

Enables the collection of the ATM OAM statistics. (*default = true*) The following statistics are controlled by this option:

atmOamRxActDeactCC	atmOamRxBadCells
atmOamRxBytes	atmOamRxFaultMgmtAIS
atmOamRxFaultMgmtCC	atmOamRxFaultMgmtLB
atmOamRxFaultMgmtRDI	atmOamRxGoodCells
atmOamTxActDeactCC	atmOamTxBytes
atmOamTxCells	atmOamTxFaultMgmtAIS
atmOamTxFaultMgmtCC	atmOamTxFaultMgmtLB
atmOamTxFaultMgmtRDU	

enableBfdStats
true/false

Enables the collection of the BFD statistics. (*default = true*) The following statistics are controlled by this option:

bfdRoutersConfigured	bfdSessionsAutoConfigured
bfdRoutersRunning	bfdAutoConfiguredSessionsUp
bfdSessionsConfigured	bfdSessionsUp

enableBgpStats
true/false

Enables the collection of the BGP statistics. (*default = true*) The following statistics are controlled by this option:

BGPTotalSessions
BGPTotalSessionsEstablished
statProtocolServerVlanDroppedFrames
statBgpSessionFlap

enableDhcpStats
true/false

Enables the collection of the DHCPv4 statistics. (*default = false*) The following statistics are controlled by this option:

dhcpV4AcksReceived	dhcpV4AddressesLearned
dhcpV4DiscoveredMessagesSent	dhcpV4EnabledInterfaces
dhcpV4NacksReceived	dhcpV4OffersReceived
dhcpV4ReleasesSent	dhcpV4RequestsSent

enableDhcpV6Stats
true/false

Enables the collection of the DHCPv6 statistics. (*default = false*) The following statistics are controlled by this option:

dhcpV6SolicitsSent	dhcpV6RepliesReceived
dhcpV6AdvertisementsReceived	dhcpV6ReleasesSent
dhcpV6RequestsSent	dhcpV6EnabledInterfaces
dhcpV6DeclinesSent	dhcpV6AddressesLearned

enableEigrpStats
true/false

Enables the collection of the EIGRP protocol statistics. (*default = true*) The following statistics are controlled by this option:

eigrpRoutersConfigured	eigrpNeighborsLearned
eigrpRoutersRunning	

enableEthernetOam
Stats
true/false

Enables the collection of the ethernet OAM protocol statistics. (*default = false*) The following statistics are controlled by this option:

ethernetOAMInformationPDUsSent	ethernetOAMInformationPDUsReceived
ethernetOAMEventNotificationPDUsReceived	ethernetOAMLoopbackControlPDUsReceived
ethernetOAMOrgPDUsReceived	ethernetOAMVariableRequestPDUsReceived
ethernetOAMVariableResponsePDUsReceived	ethernetOAMUnsupportedPDUsReceived

enableFcoeStats
true/false

Enables the collection of the protocol FCoE statistics. (*default = false*) The following statistics are controlled by this option:

fcoeFlogiSent	fcoeFlogiLsAccReceived
fcoePlogiSent	fcoePlogiLsAccReceived

fcoePlogiRequestsReceived	fcoeFlogoSent
fcoePlogoSent	fcoePlogoReceived
fcoeFdiscSent	fcoeFdiscLsAccReceived
fcoeNSRegSent	fcoeNSRegSuccessful
fcoeNxPortsEnabled	fcoeNxPortIdsAcquired
fcoeRxSharedStat1	fcoeRxSharedStat2
fpDiscoverySolicitationsSent	fpDiscoveryAdvertisementsReceived
fpKeepAlivesSent	fpClearVirtualLinksReceived

**fcoeRxSharedStat
Type1**

Only two counters are permitted, this one and the next one (*fcoeRxSharedStatType2*). Select the statistic to be assigned to this counter from these options:

statFcoelInvalidDelimiter
statFcoelInvalidFrames
statFcoelInvalidSize
statFcoeNormalSizeBadFccRc
statFcoeNormalSizeGoodFccRc
statFcoeUndersizeBadFccRc
statFcoeUndersizeGoodFccRc
statFcoevValidFrames

**fcoeRxSharedStat
Type2**

Select the statistic to be assigned to this counter from these options:

statFcoelInvalidDelimiter
statFcoelInvalidFrames
statFcoelInvalidSize
statFcoeNormalSizeBadFccRc
statFcoeNormalSizeGoodFccRc
statFcoeUndersizeBadFccRc
statFcoeUndersizeGoodFccRc
statFcoevValidFrames

**enableIcmpStats
*true/false***

Enables the collection of the ICMP statistics. (*default = true*) The following statistics are controlled by this option:

rxPingReply	rxPingRequest
txPingReply	txPingRequest

**enableIgmpStats
*true/false***

Enables the collection of the IGMP statistics. (*default = true*) The following statistics are controlled by this option:

TxIgmpFrames
RxIgmpFrames

**enableIsisStats
*true/false***

Enables the collection of the protocol ISIS statistics. (*default = true*) The following statistics are controlled by this option:

ISISSessionsConfiguredL1	ISISSessionsConfiguredL2
ISISSessionsUpL1	ISISSessionsUpL2

ISISSessionFlapL1	ISISSessionFlapL2
-------------------	-------------------

enableLacpStats
true/false

Enables the collection of the LACP protocol statistics. (*default = true*) The following statistics are controlled by this option:

lacpDURx
lacpDUTx
lacpMarkerDURx
lacpMarkerDUTx
lacpMarkerResponseRx
lacpMarkerResponseTx
lacpSessionState

enableLdpStats
true/false

Enables the collection of the protocol LDP statistics. (*default = true*) The following statistics are controlled by this option:

statLdpSessionsConfigured
statLdpSessionsUp
statLdpBasicSessionsUp
statLdpSessionFlap

enableMacSecStats
true/false

Enables the collection of the protocol MACsec statistics. (*default = true*) The following statistics are controlled by this option:

macSecValidFramesSent
macSecValidBytesSent
macSecFramesWithUnknownKeySent
macSecValidFramesReceived
macSecValidBytesReceived
macSecFramesWithUnknownKeyReceived
macSecFramesWithBadHashReceived

enableMldStats
true/false

Enables the collection of the protocol MLD statistics. (*default = true*) The following statistics are controlled by this option:

txMldFrames
rxMldFrames

enableMplsTpStats

Enables the collection of the MPLS TP protocol statistics. (*default = true*) The following statistics are controlled by this option:

statMPLSTPCccvConfigured
statMPLSTPCccvUp
statMPLSTPCccvDown

enableElmiStats

Enable the collection of ELM statistics. The following statistics are controlled by this option:

statELMIUniCConfigured
statELMIUniCRunning

statELMUniNConfigured
statELMUniNRunning
statELMUniSessionFlap
statELMISessionOperational
statELMICheckTx
statELMICheckRx
statELMIFullEnquiryTx
statELMIFullEnquiryRx
statELMIFullStatusTx
statELMIFullStatusRx
statELMIFullEnquiryContinuedTx
statELMIFullEnquiryContinuedRx
statELMIFullStatusContinuedTx
statELMIFullStatusContinuedRx
statELMIAsyncStatusTx
statELMIAsyncStatusRx

**enableNeighborSolicit
Stats
*true/false***

Enables the collection of the Neighbor Solicitation statistics. (*default = true*) The following statistics are controlled by this option:

statTxNeighborSolicits
statTxNeighborAdVERTISEMENTS
statRxNeighborSolicits
statRxNeighborAdVERTISEMENTS

**enableOspfStats
*true/false***

Enables the collection of the protocol OSPF statistics. (*default = true*) The following statistics are controlled by this option:

statOspfTotalSessions
statOspfFullNeighbors
statOspfSessionFlap

**enableOspfv3Stats
*true/false***

Enables the collection of the protocol OSPFv3 statistics. (*default = true*) The following statistics are controlled by this option:

statOspfv3SessionsConfigured
statOspfv3SessionsUp
statOspfv3SessionFlap

**enablePimsmStats
*true/false***

Enables the collection of the protocol PIM statistics. (*default = true*) The following statistics are controlled by this option:

statPimsmRoutersConfigured
statPimsmRoutersRunning
statPimsmNeighborsLearned
statPimsmSessionFlap

**enablePos
ExtendedStats
*true/false***

Enables the collection of extended PoS extended statistics, for POS cards only. (*default = true*) The following statistics are controlled by this option:

lineAis	lineBip
lineRdi	lineRei
pathAis	pathBip
pathLossOfPointer	pathPlm
pathRdi	pathRei
sectionBip	sectionLossOfFrame
sectionLossOfSignal	

**enableProtocolServer
Stats
*true/false***

Enables the collection of the protocol server statistics. (*default = true*) The following statistics are controlled by this option:

ProtocolServerTx
ProtocolServerRx
TxArpReply
TxArpRequest
TxPingRequest
RxArpReply
RxArpRequest
RxPingReply
RxPingRequest

**enablePtpStats
*true/false***

Enables the collection of the IEEE 1588 PTP statistics. (*default = true*) The following statistics are controlled by this option:

ptpAnnounceMessagesSent	ptpAnnounceMessagesReceived
ptpSyncMessagesSent	ptpSyncMessagesReceived
ptpFollowUpMessagesSent	ptpFollowUpMessagesReceived
ptpDelayRequestMessagesSent	ptpDelayRequestMessagesReceived
ptpDelayResponseMessagesSent	ptpDelayResponseMessagesReceived

**enableRsvpStats
*true/false***

Enables the collection of the RSVP protocol statistics. (*default = true*) The following statistics are controlled by this option:

RSVPIngressSubLSPsConfigured
RSVPIngressSubLSPsUp
RSVPEgressSubLSPsUp
RSVPIngressLSPsConfigured
RSVPIngressLSPsUp
RSVPEgressLSPsUp
RSVPSessionFlap

**enableStpStats
*true/false***

Enables the collection of the STP protocol statistics. (*default = true*) The following statistics are controlled by this option:

bpduFramesSent
bpduFramesReceived

**enableTemperature
SensorStats *true/false***

Enables the collection of statistics from temperature sensors. (*default = true*) The following statistics are controlled by this option:

backgroundTemperature	captureTemperature
dmaTemperature	fobBoardTemperature
fobDevice1InternalTemperature	fobPort1FpgaTemperature
frontEndTemperature	latencyTemperature
overlayTemperature	plmDevice1InternalTemperature
plmDevice2InternalTemperature	plmDevice3InternalTemperature
plmDevice4InternalTemperature	schedulerTemperature

**enableValidStats
*true / false***

If set, then *stat cget -statName* calls for statistics invalid for the port's type returns an error. If unset, then all *stat cget -statName* returns without error, but the invalid statistics have default values. (*default = 0*)

**enableVcatStats
*true/false***

Enables the collection of the VCAT per-port, per-circuit, and per-channel statistics. (*default = true*) The following statistics are available automatically when the port is in VCAT mode:

sonetCircuitState (per port)	sonetTimeslotLcasSourceState (per channel)
sonetCircuitType (per circuit)	sonetTimeslotSequenceNumberMismatch (per channel)
sonetTimeslotRsAcks (per channel)	sonetTimeslotSequenceNumber (per channel)
sonetTimeslotSlotNumber (per channel)	sonetTimeslotGfpchecErrors (per channel)
sonetTimeslotLcasSinkState (per channel)	sonetTimeslotGfpManagementFrames (per channel)
sonetTimeslotDifferentialDelay (per channel)	sonetTimeslotGfpUpimismatch (per channel)
sonetTimeslotLossOfMultiframe (per channel)	sonetTimeslotGfpGoodFramesReceived (per channel)
sonetTimeslotLossOfAlignment (per channel)	

**includeRprPayloadFcs
InCrc *true / false***

For RPR packets (where the SONET header mode is RPR), this flag indicates that the RPR payload FCS is to be included in the RPR CRC error checking. A CRC error is declared if either the RPR CRC or the Payload FCS is incorrect. (*default = true*)

lacpState

Notifies session state of LACP link, down or up.

lacpDown	LACP link is down
lacpUp	LACP link is up

mode

Sets the mode of the statistic counters. The following modes can be read:

Option	Value	Usage
statNormal	0	(<i>default</i>)
statQos	1	Reuses 8 hardware counters to count QoS packets

Option	Value	Usage
<i>statStreamTrigger</i>	2	Reuses two hardware counters: User-Defined Statistics Counters 5 and 6.
<i>statModeChecksumErrors</i>	3	Reuses 6 hardware counters to count IP, TCP, UDP checksum errors.
<i>statModeDataIntegrity</i>	4	Reuses 2 hardware counters.

Standard Options used to retrieve statistics

aggregatedGfpcHecErrors	<i>Read-only. 64-bit value.</i> Number of aggregated GFP core HEC errors detected.
aggregatedGfpeHecErrors	<i>Read-only. 64-bit value.</i> Number of aggregated GFP extension HEC errors detected.
aggregatedGfptHecErrors	<i>Read-only. 64-bit value.</i> Number of aggregated GFP type HEC errors detected.
aggregatedGfpPayloadFcsErrors	<i>Read-only. 64-bit value.</i> Number of aggregated GFP payload FCS errors detected.
aggregatedGfpManagementFrames	<i>Read-only. 64-bit value.</i> The number of aggregated GFP management frames.
aggregatedGfpUpIMismatch	<i>Read-only. 64-bit value.</i> The number of aggregated GFP UPI mismatches.
aggregatedGfpGoodFramesReceived	<i>Read-only. 64-bit value.</i> The number of aggregated GFP good frames received.
aggregatedGfpSyncState	<i>Read-only. 64-bit value.</i> The aggregated GFP sync state value.
alignmentErrors	<i>Read-only. 64-bit value.</i> Number of frames received with alignment errors on a 10/100 port.
asynchronousFramesSent	<i>Read-only. 64-bit value.</i> The number of frames sent as a part of user requests.
atmAal5BytesReceived	<i>Read-only. 64-bit value.</i> The number of AAL5 bytes received.
atmAal5BytesSent	<i>Read-only. 64-bit value.</i> The number of AAL5 bytes sent.
atmAal5CrcErrorFrames	<i>Read-only. 64-bit value.</i> The number of AAL5 frames received with CRC errors.
atmAal5FramesReceived	<i>Read-only. 64-bit value.</i> The number of AAL5 frames received.
atmAal5FramesSent	<i>Read-only. 64-bit value.</i> The number of AAL5 frames sent.
atmAal5LengthErrorFrames	<i>Read-only. 64-bit value.</i> The number of AAL5 frames received with length errors.

atmAal5TimeoutErrorFrames	<i>Read-only.</i> 64-bit value. The number of AAL5 frames received with timeout errors.
atmCellsReceived	<i>Read-only.</i> 64-bit value. The number of ATM cells received.
atmCellsSent	<i>Read-only.</i> 64-bit value. The number of ATM cells sent.
atmCorrectedHcsErrorCount	<i>Read-only.</i> 64-bit value. The number of AAL5 frames received with HCS errors that were corrected.
atmIdleCellCount	<i>Read-only.</i> 64-bit value. The number of idle ATM cells sent.
atmOamRxActDeactCC	<i>Read-only.</i> 64-bit value. Number of ATM OAM ActDeact cells transmitted.
atmOamRxBadCells	<i>Read-only.</i> 64-bit value. Number of ATM OAM bad cells received.
atmOamRxBytes	<i>Read-only.</i> 64-bit value. Number of ATM OAM bytes received.
atmOamRxFaultMgmtAIS	<i>Read-only.</i> 64-bit value. Number of ATM OAM Fault Management AIS cells received.
atmOamRxFaultMgmtCC	<i>Read-only.</i> 64-bit value. Number of ATM OAM Fault Management CC cells received.
atmOamRxFaultMgmtLB	<i>Read-only.</i> 64-bit value. Number of ATM OAM Fault Management LB cells received.
atmOamRxFaultMgmtRDI	<i>Read-only.</i> 64-bit value. Number of ATM OAM Fault Management RDI cells received.
atmOamRxGoodCells	<i>Read-only.</i> 64-bit value. Number of ATM OAM good cells received.
atmOamTxActDeactCC	<i>Read-only.</i> 64-bit value. Number of ATM OAM ActDeact cells transmitted.
atmOamTxBytes	<i>Read-only.</i> 64-bit value. Number of ATM OAM bytes transmitted.
atmOamTxCells	<i>Read-only.</i> 64-bit value. Number of ATM OAM cells transmitted.
atmOamTxFaultMgmtAIS	<i>Read-only.</i> 64-bit value. Number of ATM OAM Fault Management AIS cells transmitted.
atmOamTxFaultMgmtCC	<i>Read-only.</i> 64-bit value. Number of ATM OAM Fault Management CC cells transmitted.
atmOamTxFaultMgmtLB	<i>Read-only.</i> 64-bit value. Number of ATM OAM Fault Management LB cells transmitted.
atmOamTxFaultMgmtRDI	<i>Read-only.</i> 64-bit value. Number of ATM OAM Fault Management RDI cells transmitted.
atmScheduledCellsSent	<i>Read-only.</i> 64-bit value. The number of scheduled (non-idle) ATM cells sent.
atmUncorrectedHcsErrorCount	<i>Read-only.</i> 64-bit value. The number of AAL5 frames received with HCS errors that were not corrected.

atmUnregisteredCellsReceived	<i>Read-only.</i> 64-bit value. The number of unregistered ATM cells that were received.									
backgroundTemperature	<i>Read-only.</i> 64-bit value. Part of the OC-192 - Temperature Sensors Statistics. Temperature of the Background chip. The <i>enableTemperatureSensorsStats</i> options must be <i>true</i> for this value to be valid.									
bertAvailableSeconds	<i>Read-only.</i> 64-bit value. For BERT - the number of seconds which have occurred during Available Periods.									
bertBackgroundBlockErrorRatio	<i>Read-only.</i> For BERT: the ratio of Background Block Errors (BBEs) to the total number of blocks.									
bertBackgroundBlockErrors	<i>Read-only.</i> 64-bit value. For BERT: the number of errored blocks not occurring as part of a Severely Errored Second.									
bertBitErrorRatio	<i>Read-only.</i> For BERT: the ratio of the number of errored bits compared to the total number of bits transmitted.									
bertBitErrorsReceived	<i>Read-only.</i> 64-bit value. For BERT: the total number of bit errors received.									
bertBitErrorsSent	<i>Read-only.</i> 64-bit value. For BERT: the total number of bit errors sent.									
bertBitsReceived	<i>Read-only.</i> 64-bit value. For BERT: the total number of bits received.									
bertBitsSent	<i>Read-only.</i> 64-bit value. For BERT: the total number of bits sent.									
bertBlockErrorState	<i>Read-only.</i> For BERT: whether the link is in an available or unavailable state.									
	<table border="1"> <thead> <tr> <th>Option</th> <th>Value</th> <th>Usage</th> </tr> </thead> <tbody> <tr> <td>statBertUnavailablePeriod</td> <td>0</td> <td>Link is currently unavailable.</td> </tr> <tr> <td>statBertAvailablePeriod</td> <td>1</td> <td>Link is currently available.</td> </tr> </tbody> </table>	Option	Value	Usage	statBertUnavailablePeriod	0	Link is currently unavailable.	statBertAvailablePeriod	1	Link is currently available.
Option	Value	Usage								
statBertUnavailablePeriod	0	Link is currently unavailable.								
statBertAvailablePeriod	1	Link is currently available.								
bertDeskewPatternLock	<i>Read-only.</i> Indicates that the deskew lane has locked onto a known PRBS pattern.									
	<table border="1"> <thead> <tr> <th>Value</th> <th>Usage</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Not-locked.</td> </tr> <tr> <td>1</td> <td>Locked.</td> </tr> </tbody> </table>	Value	Usage	0	Not-locked.	1	Locked.			
Value	Usage									
0	Not-locked.									
1	Locked.									
bertElapsedTestTime	<i>Read-only.</i> 64-bit value. For BERT - the elapsed test time, expressed in seconds in the APIs.									
bertErroredBlocks	<i>Read-only.</i> 64-bit value. For BERT - the number of blocks containing at least one errored second.									
bertErrorFreeSeconds	<i>Read-only.</i> 64-bit value. For BERT - the number of seconds with no errored blocks or defects.									
bertErroredSecondRatio	<i>Read-only.</i> For BERT - (ESR); the ratio of Errored Seconds (ES) to the total seconds.									
bertErroredSeconds	<i>Read-only.</i> 64-bit value. For BERT - the number of seconds containing at least one errored block or a defect.									

bertLastServiceDisruptionTime	<i>Read-only.</i> 64-bit value. For BERT - a service disruption is the period of time during which the service is unavailable while switching rings. The SONET spec calls for this to be less than 50 ms. This value is the length of the last service disruption that occurred, expressed in milliseconds
bertMaxServiceDisruptionTime	<i>Read-only.</i> 64-bit value. For BERT - the longest service disruption that occurred, expressed in milliseconds.
bertMinServiceDisruptionTime	<i>Read-only.</i> 64-bit value. For BERT - the shortest service disruption that occurred, expressed in milliseconds.
bertMismatchedOnesRatio	<i>Read-only.</i> The number of expected ones that where received as zeroes.
bertMismatchedZerosRatio	<i>Read-only.</i> The ratio of mismatched ones to the total number of bits.
bertNumberMismatchedOnes	<i>Read-only.</i> 64-bit value. The number of expected zeroes that where received as ones.
bertNumberMismatchedZeros	<i>Read-only.</i> 64-bit value. The ratio of mismatched zeroes to the total number of bits.
bertRxDeskewBitErrors	<i>Read-only.</i> 64-bit value. The number of incorrect bits received from the deskew lane.
bertRxDeskewErroredFrames	<i>Read-only.</i> 64-bit value. The number of frames received that have at least one error.
bertRxDeskewErrorFreeFrames	<i>Read-only.</i> 64-bit value. The number of deskew frames received that have no errors.
bertRxDeskewLossOfFrame	<i>Read-only.</i> 64-bit value. The number of times that frame sync was lost and had to be re-acquired.
bertSeverelyErroredSecondRatio	<i>Read-only.</i> For BERT - (SESR); the ratio of Severely Errored Seconds (SESSs) to the total seconds.
bertServiceDisruptionCumulative	<i>Read-only.</i> 64-bit value. For BERT - the total service disruption time encountered, expressed in milliseconds.
bertStatus	<i>Read-only.</i> For BERT - the status of the receive connection. .

Option	Value	Pattern Locked
statBertNotLocked	0	None.
statBertLockedOnInvertedAllZero	1	Inverted all zeroes.
statBertLockedOnInvertedAlternatingOneZero	2	Inverted alternating one-zero.
statBertLockedOnInvertedUserDefinedPattern	3	Inverted user defined pattern.
statBertLockedOnInverted2to31powerLinearFeedbackShiftReg	4	Inverted 2**31.
statBertLockedOnInverted2to11powerLinearFeedbackShiftReg	5	Inverted 2**11.

Option	Value	Pattern Locked
statBertLockedOnInverted 2to15powerLinearFeedbackShiftReg	6	Inverted 2**15.
statBertLockedOnInverted 2to20powerLinearFeedbackShiftReg	7	Inverted 2**20.
statBertLockedOnInverted 2to23powerLinearFeedbackShiftReg	8	Inverted 2**23.
statBertLockedOnAllZero	9	All zeroes.
statBertLockedOnAlternatingAllZero	10	Alternating one-zero.
statBertLockedOnAlternatingOneZero	11	User defined pattern.
statBertLockedOn 2to31powerLinearFeedbackShiftReg	12	2**31.
statBertLockedOn 2to11powerLinearFeedbackShiftReg	13	2**11.
statBertLockedOn 2to15powerLinearFeedbackShiftReg	14	2**15.
statBertLockedOn 2to20powerLinearFeedbackShiftReg	15	2**20.
statBertLockedOn 2to23powerLinearFeedbackShiftReg	16	2**23.

bertTimeSinceLast Error	<i>Read-only.</i> 64-bit value. The elapsed time since the last receive error was detected, expressed in nano-seconds.
bertTransmitDuration	<i>Read-only.</i> The transmit duration time when port is in BERT mode.
bertTriggerCount	<i>Read-only.</i> 64-bit value. The number of triggers generated.
bertTxDeskewBitErrors	<i>Read-only.</i> 64-bit value. The number of bit errors inserted into the transmitted deskew lane.
bertTxDeskewErrored Frames	<i>Read-only.</i> 64-bit value. The number of transmitted deskew frames with at least one error inserted.
bertTxDeskewError FreeFrames	<i>Read-only.</i> 64-bit value. The number of transmitted deskew frames with at no errors inserted.
bertUnavailable Seconds	<i>Read-only.</i> 64-bit value. For BERT - the number of seconds which have occurred during Unavailable Periods.
bertUnframed DetectedLineRate	<i>Read-only.</i> 64-bit value. For unframed BERT - the detected line rate, in bps.
bertUnframed OutputSignalStrength	<i>Read-only.</i> DOUBLE value. For unframed BERT - the output signal strength, in db.
bgpTotalSessions	<i>Read-only.</i> 64-bit value. For BGP4 - the number of BGP4 sessions that were configured. The <i>enableProtocolServer</i> and <i>enableBgpStats</i> options must be <i>true</i> for this value to be valid.
bgpTotalSessions Established	<i>Read-only.</i> 64-bit value. (For BGP4) The number of configured BGP4 sessions that established adjacencies. The <i>enableProtocolServer</i> and <i>enableBgpStats</i> options must be <i>true</i> for this value to be valid.

bitsReceived	<i>Read-only.</i> 64-bit value. Number of bits received.									
bitsSent	<i>Read-only.</i> 64-bit value. Number of bits transmitted.									
bpduFramesReceived	<i>Read-only.</i> 64-bit value. Number of bridging protocol data units received.									
bpduFramesSent	<i>Read-only.</i> 64-bit value. Number of bridging protocol data units sent.									
bytesFrom Application	<i>Read-only.</i> 64-bit value. On the stream extraction module, number of bytes received from the application on either port 2 or port 3.									
bytesReceived	<i>Read-only.</i> 64-bit value. Number of bytes received.									
bytesSent	<i>Read-only.</i> 64-bit value. Number of bytes transmitted.									
captureFilter	<i>Read-only.</i> 64-bit value. Number of frames received meeting the capture filter criteria set up using filter command. This counter is available when stat mode is set to statNormal.									
captureState	<i>Read-only.</i> Reflects the current state of capture. The following states can be read:									
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Option</th> <th style="text-align: center;">Value</th> <th style="text-align: center;">Usage</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;"><i>statIdle</i></td> <td style="text-align: center;">0</td> <td>capture stopped</td> </tr> <tr> <td style="text-align: center;"><i>statActive</i></td> <td style="text-align: center;">1</td> <td>port currently capturing</td> </tr> </tbody> </table>	Option	Value	Usage	<i>statIdle</i>	0	capture stopped	<i>statActive</i>	1	port currently capturing
Option	Value	Usage								
<i>statIdle</i>	0	capture stopped								
<i>statActive</i>	1	port currently capturing								
captureTemperature	<i>Read-only.</i> 64-bit value. Part of the OC-192 - Temperature Sensors Statistics. Temperature of the Capture chip. The enableTemperatureSensorsStats options must be true for this value to be valid.									
captureTrigger	<i>Read-only.</i> 64-bit value. Number of frames received meeting the capture trigger criteria set up using filter command. This counter is available when stat mode is set to statNormal.									
cdlErrorFrames Received	<i>Read-only.</i> 64-bit value. Number of errored CDL frames received.									
cdlGoodFrames Received	<i>Read-only.</i> 64-bit value. Number of good CDL frames received.									
codingErrorFrames Received	<i>Read-only.</i> 64-bit value. The number of frames received with coding errors.									
collisionFrames	<i>Read-only.</i> 64-bit value. Number of frames received with collisions.									
collisions	<i>Read-only.</i> 64-bit value. Number of collisions.									
customOrderedSet Received	<i>Read-only.</i> 64-bit value. The number of remote ordered sets received. Ordered sets are part of Link Fault Signaling, and can be configured with Link Fault Signaling.									
customOrderedSetSent	<i>Read-only.</i> 64-bit value. The number of custom ordered sets sent. Ordered sets are part of Link Fault Signaling, and can be configured with Link Fault Signaling.									
dataIntegrityErrors	<i>Read-only.</i> 64-bit value. Number of frames that have data integrity error. (Not available when port is in PRBS mode.)									

dataIntegrityFrames	<i>Read-only.</i> 64-bit value. Number of frames that match data integrity signature. (Not available when port is in PRBS mode.)
dcbxIscsiTlvsSent	<i>Read-only.</i> Number of DCBX ISCSI TLVs sent.
dcbxIscsiTlvsReceived	<i>Read-only.</i> Number of DCBX ISCSI TLVs received.
dccBytesReceived	<i>Read-only.</i> 64-bit value. Number of DCC bytes received.
dccBytesSent	<i>Read-only.</i> 64-bit value. Number of DCC bytes sent.
dccCrcErrorsReceived	<i>Read-only.</i> 64-bit value. Number of DCC CRC errors received.
dccFramesReceived	<i>Read-only.</i> 64-bit value. Number of DCC frames received.
dccFramesSent	<i>Read-only.</i> 64-bit value. Number of DCC frames sent.
dccFramingErrorsReceived	<i>Read-only.</i> 64-bit value. Number of DCC framing errors received.
dhcpV4AcksReceived	<i>Read-only.</i> 64-bit value. The number of ACK messages received.
dhcpV4AddressesLearned	<i>Read-only.</i> 64-bit value. The number of address learned.
dhcpV4DiscoveredMessagesSent	<i>Read-only.</i> 64-bit value. The number of Discovered messages sent.
dhcpV4EnabledInterfaces	<i>Read-only.</i> 64-bit value. The number of enabled interfaces.
dhcpV4NacksReceived	<i>Read-only.</i> 64-bit value. The number of NACK messages received.
dhcpV4OffersReceived	<i>Read-only.</i> 64-bit value. The number of Offer messages received.
dhcpV4ReleasesSent	<i>Read-only.</i> 64-bit value. The number of Release messages sent.
dhcpV4RequestsSent	<i>Read-only.</i> 64-bit value. The number of Request messages sent.
dmaTemperature	<i>Read-only.</i> 64-bit value. Part of the OC-192 - Temperature Sensors Statistics. Temperature of the DMA chip. The <i>enableTemperatureSensorsStats</i> options must be <i>true</i> for this value to be valid.
dribbleErrors	<i>Read-only.</i> 64-bit value. Number of frames received with dribble errors on a 10/100 port.
droppedFrames	<i>Read-only.</i> 64-bit value. The number of dropped frames.
duplexMode	<i>Read-only.</i> The duplex mode configured for the port. The following states can be read:

Value	Usage
0	half duplex
1	full duplex

eErrorCharacterFramesReceived	<i>Read-only.</i> 64-bit value. The number of frames received with DUT labeled errors received.
egressDroppedFrames	<i>Read-only.</i> 64-bit value. The number of frames that get dropped before they are transmitted.
ethernetCrc	<i>Read-only.</i> The ethernet CRC for ATM cards, the CRC counter represents AAL5 CRCs.
ethernetOAMInformationPDUsSent	<i>Read-only.</i> The number of ethernet OAM Information PDUs sent.
ethernetOAMInformationPDUsReceived	<i>Read-only.</i> The number of ethernet OAM Information PDUs received.
ethernetOAMEventNotificationPDUsReceived	<i>Read-only.</i> The number of ethernet OAM Event Notification PDUs received.
ethernetOAMLoopbackControlPDUsReceived	<i>Read-only.</i> The number of ethernet OAM Loopback Control PDUs received.
ethernetOAMOrgPDUsReceived	<i>Read-only.</i> The number of ethernet OAM Organization Specific PDUs received.
ethernetOAMVariableRequestPDUsReceived	<i>Read-only.</i> The number of ethernet OAM Variable Request PDUs received.
ethernetOAMVariableResponsePDUsReceived	<i>Read-only.</i> The number of ethernet OAM Variable Response PDUs received.
ethernetOAMUnsupportedPDUsReceived	<i>Read-only.</i> The number of ethernet OAM Unsupported PDUs received.
excessiveCollisionFrames	<i>Read-only.</i> 64-bit value. Number of frames received with excessive collisions.
fcoeFdiscLsAccReceived	<i>Read-only.</i> FCoE Discovery Link Service Accept received.
fcoeFdiscSent	<i>Read-only.</i> FCoE Discovery sent.
fcoeFlogiLsAccReceived	<i>Read-only.</i> FCoE Fabric Login Link Service Accept received.
fcoeFlogiSent	<i>Read-only.</i> FCoE Fabric Login sent.
fcoeFlogoSent	<i>Read-only.</i> FCoE Fabric Logout sent.
fcoeNxPortIdsAcquired	<i>Read-only.</i> FCOE Nx Port IDs Acquired
fcoeNxPortsEnabled	<i>Read-only.</i> FCOE Nx Ports Enabled
fcoeNSRegSent	<i>Read-only.</i> FCOE Name Server Registration sent
fcoeNSRegSuccessful	<i>Read-only.</i> FCOE Name Server Registration successful

fcoePlogiLsAccReceived	<i>Read-only.</i> FCOE Port Login Link Service Accept received
fcoePlogiRequests Received	<i>Read-only.</i> FCOE Port Login Requests received
fcoePlogiSent	<i>Read-only.</i> FCOE Port Login sent
fcoePlogoReceived	<i>Read-only.</i> FCOE Port Logout received
fcoePlogoSent	<i>Read-only.</i> FCOE Port Logout sent
fcoeRxSharedStat1	<i>Read-only.</i> The requested FCoE variable is stored here.
fcoeRxSharedStat2	<i>Read-only.</i> The requested FCoE variable is stored here.
fcsErrors	<i>Read-only.</i> 64-bit value. Number of frames received with FCS errors.
fecCorrected0sCount	<i>Read-only.</i> 64-bit value. Number of 0 errors (1s changed to 0s) that have been corrected.
fecCorrected1sCount	<i>Read-only.</i> 64-bit value. Number of 1 errors (0s changed to 1s) that have been corrected.
fecCorrectedBitsCount	<i>Read-only.</i> 64-bit value. Number of flipped bits errors (0s changed to 1s and vice versa) that have been corrected.
fecCorrectedBytes Count	<i>Read-only.</i> 64-bit value. Number of bytes that have had errors corrected.
fecUncorrectable SubrowCount	<i>Read-only.</i> 64-bit value. Number of subrows that have had uncorrectable errors.
fipDiscoverySolicitationsSent	<i>Read-only.</i> Number of FIP Discovery Solicitations that have been sent.
fipDiscoveryAdVERTISEmentsReceived	<i>Read-only.</i> Number of FIP Discovery Advertisements that have been received.
fipKeepAlivesSent	<i>Read-only.</i> Number of FIP Keep Alives that have been sent.
fipClearVirtualLinks Received	<i>Read-only.</i> Number of FIP Clear Virtual Links that have been received.
flowControlFrames	<i>Read-only.</i> 64-bit value. Number of flow control frames received.
fobBoardTemperature	<i>Read-only.</i> 64-bit value. Part of the OC-192 - Temperature Sensors Stats. The temperature of the board of the Fiber optic module. The <i>enableTemperatureSensorsStats</i> options must be <i>true</i> for this value to be valid.
fobDeviceInternal Temperature	<i>Read-only.</i> 64-bit value. Part of the OC-192 - Temperature Sensors Stats. The temperature of the FPGA on the Fiber optic module. The <i>enableTemperatureSensorsStats</i> options must be <i>true</i> for this value to be valid.
fobPort1Fpga Temperature	<i>Read-only.</i> 64-bit value. Part of the OC-192 - Temperature Sensors Stats. The temperature next to port 1 on the Fiber optic module. The <i>enableTemperatureSensorsStats</i> options must be <i>true</i> for this value to be valid.

fobPort2FpgaTemperature	<i>Read-only.</i> 64-bit value. Part of the OC-192 - Temperature Sensors Stats. The temperature next to port 2 on the Fiber optic module.									
fragments	<i>Read-only.</i> 64-bit value. Number of fragmented frames received.									
framerAbort	<i>Read-only.</i> 64-bit value.									
framerFCSErrors	<i>Read-only.</i> 64-bit value.									
framerMaxLength	<i>Read-only.</i> 64-bit value.									
framerMinLength	<i>Read-only.</i> 64-bit value.									
framesReceived	<i>Read-only.</i> 64-bit value. Number of frames received.									
framesSent	<i>Read-only.</i> 64-bit value. Number of frames transmitted.									
frontEndTemperature	<i>Read-only.</i> 64-bit value. Part of the OC-192 - Temperature Sensors Statistics. Temperature of the Front End chip. The <i>enableTemperatureSensorsStats</i> options must be <i>true</i> for this value to be valid.									
gfpIdleFrames	<i>Read-only.</i> 64-bit value. Number of GFP idle frames transmitted.									
gfpSyncHuntTransitions	<i>Read-only.</i> 64-bit value. The number of Sync/Hunt state transition frames received.									
gfpeHecErrors	<i>Read-only.</i> 64-bit value. Number of GFP extension header HEC errors detected.									
gfpPayloadFcsErrors	<i>Read-only.</i> 64-bit value. Number of payload FCS errors detected.									
gfpRxBandwidth	<i>Read-only.</i> 64-bit value. The measured receive GFP bandwidth, in Mbps.									
gfpSyncState	<i>Read-only.</i> 64-bit value. The GFP sync state value.									
gfptHecErrors	<i>Read-only.</i> 64-bit value. Number of GFP type header HEC errors detected.									
inputSignalStrength	<i>Read-only.</i> Monitors receive optical input power.									
insertionState	<i>Read-only.</i> The current state of link fault insertion.									
<table border="1"><thead><tr><th>Option</th><th>Value</th><th>Usage</th></tr></thead><tbody><tr><td><i>linkFaultInsertionIdle</i></td><td>0</td><td>No error insertion.</td></tr><tr><td><i>linkFaultInsertionInserting</i></td><td>1</td><td>In the process of inserting link faults.</td></tr></tbody></table>		Option	Value	Usage	<i>linkFaultInsertionIdle</i>	0	No error insertion.	<i>linkFaultInsertionInserting</i>	1	In the process of inserting link faults.
Option	Value	Usage								
<i>linkFaultInsertionIdle</i>	0	No error insertion.								
<i>linkFaultInsertionInserting</i>	1	In the process of inserting link faults.								
ipChecksumErrors	<i>Read-only.</i> 64-bit value. Number of frames transmitted.									
ipPackets	<i>Read-only.</i> 64-bit value. Number of frames transmitted.									
lateCollisions	<i>Read-only.</i> 64-bit value. Number of frames received with late collisions.									
latencyTemperature	<i>Read-only.</i> 64-bit value. Part of the OC-192 - Temperature Sensors Statistics. Temperature of the Latency chip. The <i>enableTemperatureSensorsStats</i> options must be <i>true</i> for this value to be valid.									

lineAis

Read-only. A flag indicating whether any Line Alarm Indication Signal have been received on an OC port for Packet over Sonet interfaces.

Value	Usage
0	no errors
1	alarm
2	not applicable

The `enablePosExtendedStats` options must be *true* for this value to be valid.

lineAisAlarmSecs

Read-only. 64-bit value. A count of the seconds during which (at any point during the second) at least one Line layer AIS defect was present.

lineBip

Read-only. 64-bit value. Number of Line Bit Interleaved Parity errors received on OC ports for POS interfaces. The `enablePosExtendedStats` options must be *true* for this value to be valid.

lineBipErroredSecs

Read-only. 64-bit value. A count of the seconds during which (at any point during the second) at least one Line layer BIP was detected.

lineRdi

Read-only. 64-bit value. A flag indicating whether any Line Remote Defect Indicators (former FERF: Far End Receive Failure) have been received on an OC ports for Packet over Sonet interfaces. Contains value after the class method `stat get statAllStats` is used:

Value	Usage
0	no errors
1	alarm
2	not applicable

The `enablePosExtendedStats` options must be *true* for this value to be valid.

lineRdiUnavailableSecs

Read-only. 64-bit value. A count of the seconds during which the line is considered unavailable at the far end.

lineRei

Read-only. 64-bit value. Number of Line Remote Error Indications (former FEBE: Far End Block Error) received on OC ports for Packet over Sonet interfaces. The `enablePosExtendedStats` options must be *true* for this value to be valid.

lineReiErroredSecs

Read-only. 64-bit value. A count of the seconds during which at least one line BIP error was reported by the far end.

lineSpeed

Read-only. The speed configured for the port.

link

Read-only. The state of the link. The following states can be read from the port:

Option	Value	Usage
<code>linkDown</code>	0	The link on the port is down. This may be because there is no cable connected to the port or the link on the destination port may be down. The LED on the card is off when the link is down. (<i>default</i>)
<code>linkUp</code>	1	the link is up indicated by green LED on the card.

Option	Value	Usage
<i>linkLoopback</i>	2	the port has been set to loopback mode. The LED on the card is off in this mode.
<i>miiWrite</i>	3	the link goes into this state when the configuration of 10/100 port is being written to hardware (<i>applicable to 10/100 only</i>)
<i>restartAuto</i>	4	restarts the auto-negotiation process
<i>autoNegotiating</i>	5	the link is in currently executing the auto-negotiation process
<i>miiFail</i>	6	failed to write into memory for 10/100 ports (<i>applicable to 10/100 only</i>)
<i>noTransceiver</i>	7	No external transceiver detected on Ixia Mii or Rmii port.
<i>invalidAddress</i>	8	No PHY detected at the selected address.
<i>readLinkPartner</i>	9	Auto negotiation state in negotiation process. This is an intermediate state and should be used for informational purposes only.
<i>noLinkPartner</i>	10	Auto negotiation state in negotiation process. No link partner was found. This is an intermediate state and should be used for informational purposes only.
<i>restartAutoEnd</i>	11	Auto negotiation state in negotiation process. This is an intermediate state and should be used for informational purposes only.
<i>fpgaDownloadFail</i>	12	Fpga download failure. Port is not be usable.
<i>noGbicModule</i>	13	No GBIC module detected on Ixia GBic port.
<i>fifoReset</i>	14	State in board initialization process. This is an intermediate state and should be used for informational purposes only.
<i>fifoResetComplete</i>	15	State in board initialization process. This is an intermediate state and should be used for informational purposes only.
<i>pppOff</i>	16	PPP is disabled. PPP control packets are ignored; PPP link negotiation is not performed. Does not mean the link is unusable; it may, for instance, be configured for Cisco/HDLC and traffic (non-PPP) may still flow.
<i>pppUp</i>	17	The fully operational state when PPP is enabled. PPP link negotiation has successfully completed and the link is available for normal data traffic.
<i>pppDown</i>	18	The non-operational state when PPP is enabled. PPP link negotiation has failed or the link has been administratively disabled.
<i>pppInit</i>	19	PPP link negotiation state. <i>This is an intermediate state and should be used for informational purposes only.</i> Initialization state at the start of the negotiation process.
<i>pppWaitForOpen</i>	20	PPP link negotiation state: Waiting for indication from PPP controller that auto-negotiation and related PPP control packet transfers can proceed. <i>This is an intermediate state and should be used for informational purposes only.</i>

Option	Value	Usage
<i>pppAutoNegotiate</i>	21	PPP link negotiation state: In process of exchanging PPP control packets (for example, LCP and IPCP) to negotiate link parameters. <i>This is an intermediate state and should be used for informational purposes only.</i>
<i>pppClose</i>	22	PPP link negotiation state: The PPP session has been terminated. All data traffic stops.
<i>pppConnect</i>	23	PPP link negotiation state: Negotiation has successfully completed; the peers are logically connected. Normal data traffic may flow once the pppUp state is reached. <i>This is an intermediate state and should be used for informational purposes only.</i>
<i>lossOfSignal</i>	25	Physical link is down. (for example, loss of signal, loss of frame)
<i>lossOfFramePppDisabled</i>	26	PPP link negotiation state: Physical link has gone down and PPP negotiation has been stopped.
<i>stateMachineFailure</i>	27	Communication with the local processor has failed. Check Server display and log for possible failure.
<i>pppRestartNegotiation</i>	28	PPP link negotiation state, following explicit request to restart negotiation process: this state indicates response to request. <i>This is an intermediate state and should be used for informational purposes only.</i>
<i>pppRestartInit</i>	29	PPP link negotiation state, following explicit request to restart negotiation process: the link has or is brought down to begin a new negotiation cycle. <i>This is an intermediate state and should be used for informational purposes only.</i>
<i>pppRestartWaitForOpen</i>	30	PPP link negotiation state, following explicit request to restart negotiation process: Waiting for indication from PPP controller that current connection is already down or is in process of being shut down. <i>This is an intermediate state and should be used for informational purposes only.</i>
<i>pppRestartWaitForClose</i>	31	PPP link negotiation state, following explicit request to restart negotiation process: Waiting for indication from PPP controller that shut down of current connection has completed. <i>This is an intermediate state and should be used for informational purposes only.</i>
<i>pppRestartFinish</i>	32	PPP link negotiation state, following explicit request to restart negotiation process: Preparation for restart completed; ready to begin normal cycle again. <i>This is an intermediate state and should be used for informational purposes only.</i>
<i>localProcessorDown</i>	33	local processor boot failure
<i>sublayerUnlock</i>	41	Sublayer unlock.
<i>demoMode</i>	42	Server is in demo mode.
<i>waitingForFpgaDownload</i>	43	Port is waiting for FPGA (Field Programmable Gate Array) programming to be downloaded to port.
<i>lossOfCell</i>	44	ATM cell loss.
<i>noXFPMModule</i>	45	No XFP module is installed.
<i>moduleNotReady</i>	46	The XFP interface has reported not ready.
<i>noX2Module</i>	48	No X2 module is installed.

Option	Value	Usage
<i>lossOfPointer</i>	49	Loss of pointer.
<i>lossOfAlignment</i>	50	Loss of alignment.
<i>lossOfMultiframe</i>	51	Loss of multiframe.
<i>gfpOutOfSync</i>	52	GFP out of sync.
<i>lcasSequenceMismatch</i>	53	Lcas sequence mismatch.
<i>etherenOamLoopback</i>	54	Ethernet OAM Loopback state

linkFaultState

Read-only. The current detected link fault state for the port.

Option	Value	Usage
<i>noLinkFault</i>	0	No link fault detected.
<i>localLinkFault</i>	1	A local link fault has been detected.
<i>remoteFault</i>	2	A remote link fault has been detected.

localFaults

Read-only. 64-bit value. The number of local link faults detected.

localOrderedSetReceived

Read-only. 64-bit value. The number of local ordered sets received. Ordered sets are part of Link Fault Signaling.

localOrderedSetSent

Read-only. 64-bit value. The number of local ordered sets sent. Ordered sets are part of Link Fault Signaling.

misdirectedPacketsReceived

Read-only. 64-bit value. The number of misdirected packets received.

monitorBytesFromPort2

Read-only. 64-bit value. On the stream extraction module, the number of bytes from port two to the monitor port.

monitorBytesFromPort3

Read-only. 64-bit value. On the stream extraction module, the number of bytes from port three to the monitor port.

monitorPacketsFromPort2

Read-only. 64-bit value. On the stream extraction module, the number of packets from port two to the monitor port.

monitorPacketsFromPort3

Read-only. 64-bit value. On the stream extraction module, the number of packets from port three to the monitor port.

overlayTemperature

Read-only. 64-bit value. Part of the OC-192 - Temperature Sensors Statistics. Temperature of the Overlay chip. The *enableTemperatureSensorsStats* options must be *true* for this value to be valid.

oversize

Read-only. 64-bit value. Number of oversized frames received (greater than 1518 bytes).

oversizeAndCrcErrors

Read-only. 64-bit value. Only available for Gigabit modules. Number of frames received with oversize and CRC errors.

packetsFromApplication

Read-only. 64-bit value. On the stream extraction module, number of packets received from the application on either port 2 or port 3.

**packetsSkippedIn
PacketGroupMode**

Read-only. 64-bit value. The number of packets which were not assigned to a packet group. This can occur if the packet contains the anticipated packet group signature, but is too short to hold the group ID.

pathAis

Read-only. A flag indicating whether any Path Alarm Indication Signals have been received on an OC ports for Packet over Sonet interfaces. Contains a value after the class method **stat get statAllStats** is used:

Option	Value	Usage
	0	no errors
	1	alarm
	2	not applicable

The *enablePosExtendedStats* options must be *true* for this value to be valid.

pathAisErroredSecs

Read-only. 64-bit value. A count of the seconds during which (at any point during the second) at least one Path AIS error was detected.

pathAisUnavailableSecs

Read-only. 64-bit value. A count of the seconds during which the STS path was considered unavailable.

pathBip

Read-only. 64-bit value. Number of Path Bit Interleaved Parity errors received on OC ports for Packet over Sonet interfaces. The *enablePosExtendedStats* options must be *true* for this value to be valid.

pathBipErroredSecs

Read-only. 64-bit value. A count of the seconds during which (at any point during the second) at least one Path BIP error was detected.

pathLossOfPointer

Read-only. A flag indicating whether any Path LOP indications have been received on an OC ports for Packet over Sonet interfaces. Contains a value after the class method **stat get statAllStats** is used:

Option	Value	Usage
	0	no errors
	1	alarm
	2	not applicable

The *enablePosExtendedStats* options must be *true* for this value to be valid.

pathPlm

Read-only. A flag indicating whether any Path Label Mismatch indications have been received on an OC ports for Packet over Sonet interfaces. Contains a value after the class method **stat get statAllStats** is used:

Option	Value	Usage
	0	no errors
	1	alarm
	2	not applicable

The *enablePosExtendedStats* options must be *true* for this value to be valid.

pathRdi

Read-only. A flag indicating whether any Path Remote Defect Indicators (former FERF: Far End Receive Failure) have been received on an OC ports for Packet

over Sonet interfaces. Contains a value after the class method **stat get statAllStats** is used:

Option	Value	Usage
	0	no errors
	1	alarm
	2	not applicable

The *enablePosExtendedStats* options must be *true* for this value to be valid.

pathRdiUnavailableSecs

Read-only. 64-bit value. A count of the seconds during which the STS path was considered unavailable at the far end.

pathRei

Read-only. 64-bit value. Number of Path Remote Error Indications (former FEBE : Far End Block Error) received on OC ports for Packet over Sonet interfaces. The *enablePosExtendedStats* options must be *true* for this value to be valid.

pathReiErroredSecs

Read-only. 64-bit value. A count of the seconds during which (at any point during the second) at least one STS Path error was reported by the far end.

pauseAcknowledge

Read-only. 64-bit value. For 10Gbe: the number of received pause acknowledge messages.

pauseEndFrames

Read-only. 64-bit value. For 10Gbe: the number of received pause end frame messages.

pauseOverwrite

Read-only. 64-bit value. For 10Gbe: the number of pause frames received while transmit was paused with a quanta not equal to 0.

pauseState

Read-only. Reflects whether the port is in pause transmit mode. The following states can be read:

Option	Value	Usage
<i>statIdle</i>	0	transmit pause not enabled
<i>statActive</i>	1	transmit pause enabled

pcsSyncErrorsReceived

Read-only. The number of 64B/66B blocks received with a sync header that does not have a valid value of either *01* (data) or *10* (control).

pcsIllegalCodesReceived

Read-only. The number of 64B/66B control blocks received with a block type field that is not among one of the following valid types of 64B/66B Block Formats: 0x1E, 0x78, 0x4B, 0x87, 0x99, 0xAA, 0xB4, 0xCC, 0xD2, 0xE1, 0xFF.

pcsRemoteFaultsReceived

Read-only. The number of Remote Fault sequence ordered sets received by the test port.

pcsLocalFaultsReceived

Read-only. The number of Local Fault sequence ordered sets received by the test port.

pcsIllegalOrderedSetReceived

Read-only. The number of 64B/66B control blocks received with a block type field of 0x4B for Ordered Sets, and the remainder of the block does not match that of valid ordered set codes (for local fault or remote fault).

pcsIllegalIdleReceived	<i>Read-only.</i> The number of 64B/66B control blocks received with a block type field of 0x1E, and the remainder of the block does not contain all valid idle control codes.
pcsIllegalSofReceived	<i>Read-only.</i> The number of 64B/66B control blocks received with a block type field of 0x78 for a Start code, and the remainder of the block does not match that of a valid preamble (0x55_55_55_55_55_D5). If the port has programmable preamble mode enabled, the remainder of the block is allowed to have any value, and so no blocks will be counted as Illegal SOF.
pcsOutOfOrderSofReceived	<i>Read-only.</i> The number of SOF control blocks received while in the middle of a frame. In other words, a 64B/66B SOF control block was received (block type field = 0x78) to start a frame, possibly followed by additional Data blocks, followed by another SOF block prior to having received an EOF control block to terminate the frame.
pcsOutOfOrderEofReceived	<i>Read-only.</i> The number of EOF control blocks received while not in the middle of a frame. In other words, an EOF control block was received without having received an SOF control block to start the frame.
pcsOutOfOrderDataReceived	<i>Read-only.</i> The number of Data blocks received while not in the middle of a frame. In other words, a Data block was received without having received an SOF control block to start the frame.
pcsOutOfOrderOrderedSetReceived	<i>Read-only.</i> The number of Ordered Set blocks received while in the middle of a frame. In other words, a 64B/66B SOF control block was received (block type field = 0x78) to start a frame, possibly followed by additional Data blocks, followed by an ordered set block prior to having received an EOF control block to terminate the frame.
plmDevice1InternalTemperature	<i>Read-only. 64-bit value.</i> Part of the OC-192 - Temperature Sensors Statistics. Temperature of the PLM measuring device #1 chip. The <i>enableTemperatureSensorsStats</i> options must be <i>true</i> for this value to be valid.
plmDevice2InternalTemperature	<i>Read-only. 64-bit value.</i> Part of the OC-192 - Temperature Sensors Statistics. Temperature of the PLM measuring device #2 chip. The <i>enableTemperatureSensorsStats</i> options must be <i>true</i> for this value to be valid.
plmDevice3InternalTemperature	<i>Read-only. 64-bit value.</i> Part of the OC-192 - Temperature Sensors Statistics. Temperature of the PLM measuring device #3 chip. The <i>enableTemperatureSensorsStats</i> options must be <i>true</i> for this value to be valid.
poeActiveInput	<i>Read-only. 64-bit value.</i> Displays the type of PSE in use, Alt. A or Alt B
poeAmplitudeArmStatus	<i>Read-only.</i> The state of <i>poeSignalAcquisition</i> amplitude measurement arming; <i>true</i> = armed and <i>false</i> = not armed.
poeAmplitudeDoneStatus	<i>Read-only.</i> The state of <i>poeSignalAcquisition</i> amplitude measurement; <i>true</i> = measurement has been taken and <i>false</i> = not taken.
poeAutocalibration	<i>Read-only.</i> The stage in the port diagnostic test.
poeInputCurrent	<i>Read-only. Floating point value.</i> The port's input current.

poeInputPower	<i>Read-only.</i> Floating point value. The port's input power.																																	
poeInputVoltage	<i>Read-only.</i> Floating point value. The port's input voltage.																																	
poeStatus	<i>Read-only.</i> The state of the Power Over Ethernet port. Possible states:																																	
	<table border="1"> <thead> <tr> <th>Option</th> <th>Value</th> <th>Usage</th> </tr> </thead> <tbody> <tr><td><i>statPoeNoOperation</i></td><td>0</td><td>POE no operation</td></tr> <tr><td><i>statPoeDetect</i></td><td>1</td><td>POE detect</td></tr> <tr><td><i>statPoeClassify</i></td><td>2</td><td>POE classify</td></tr> <tr><td><i>statPoeReady</i></td><td>3</td><td>POE ready</td></tr> <tr><td><i>statPoeOperate</i></td><td>4</td><td>POE operate</td></tr> <tr><td><i>statPoePulse</i></td><td>5</td><td>POE pulse</td></tr> <tr><td><i>statPoeOff</i></td><td>6</td><td>POE off</td></tr> <tr><td><i>statPoeIdle</i></td><td>7</td><td>POE idle</td></tr> <tr><td><i>statPoeError</i></td><td>8</td><td>POE error</td></tr> <tr><td><i>statPoeShutdown</i></td><td>9</td><td>POE shutdown</td></tr> </tbody> </table>	Option	Value	Usage	<i>statPoeNoOperation</i>	0	POE no operation	<i>statPoeDetect</i>	1	POE detect	<i>statPoeClassify</i>	2	POE classify	<i>statPoeReady</i>	3	POE ready	<i>statPoeOperate</i>	4	POE operate	<i>statPoePulse</i>	5	POE pulse	<i>statPoeOff</i>	6	POE off	<i>statPoeIdle</i>	7	POE idle	<i>statPoeError</i>	8	POE error	<i>statPoeShutdown</i>	9	POE shutdown
Option	Value	Usage																																
<i>statPoeNoOperation</i>	0	POE no operation																																
<i>statPoeDetect</i>	1	POE detect																																
<i>statPoeClassify</i>	2	POE classify																																
<i>statPoeReady</i>	3	POE ready																																
<i>statPoeOperate</i>	4	POE operate																																
<i>statPoePulse</i>	5	POE pulse																																
<i>statPoeOff</i>	6	POE off																																
<i>statPoeIdle</i>	7	POE idle																																
<i>statPoeError</i>	8	POE error																																
<i>statPoeShutdown</i>	9	POE shutdown																																
poeTemperature	<i>Read-only.</i> The temperature of the PoE port, in Celsius.																																	
poeTimeArmStatus	<i>Read-only.</i> The state of <i>poeSignalAcquisition</i> time measurement arming; <i>true</i> = armed and <i>false</i> = not armed.																																	
poeTimeDoneStatus	<i>Read-only.</i> The state of <i>poeSignalAcquisition</i> time measurement trigger; <i>true</i> = triggered and <i>false</i> = not triggered.																																	
poeTriggerAmplitude DCamps	<i>Read-only.</i> Floating point value. The measured DC amps value from a triggered event set up in <i>poeSignalAcquisition</i> .																																	
poeTriggerAmplitude DCVolts	<i>Read-only.</i> Floating point value. The measured DC volts value from a triggered event set up in <i>poeSignalAcquisition</i> .																																	
poeTriggerTime	<i>Read-only.</i> Floating point value. The measured time value from a triggered event set up in <i>poeSignalAcquisition</i> .																																	
portCPUBBytesReceived	<i>Read-only.</i> 64-bit value. Number of bytes that are received by port CPU.																																	
portCPUFramesReceived	<i>Read-only.</i> 64-bit value. Number of frames that are received by port CPU.																																	
portCPUFramesSent	<i>Read-only.</i> 64-bit value. The number of frames originating from the port's CPU rather than the stream engine.																																	
portCpuIngressDroppedFrames	<i>Read-only.</i> 64-bit value. The number of frames that dropped while coming to the port cpu.																																	
portCpuStatus	<i>Read-only.</i> The state of the port's CPU. One of																																	
	<table border="1"> <thead> <tr> <th>Option</th> <th>Value</th> <th>Usage</th> </tr> </thead> <tbody> <tr><td><i>statCpuNotPresent</i></td><td>0</td><td>No CPU is present on this port.</td></tr> <tr><td><i>statCpuNotReady</i></td><td>1</td><td>The CPU is not ready.</td></tr> <tr><td><i>statCpuReady</i></td><td>2</td><td>The CPU is ready.</td></tr> </tbody> </table>	Option	Value	Usage	<i>statCpuNotPresent</i>	0	No CPU is present on this port.	<i>statCpuNotReady</i>	1	The CPU is not ready.	<i>statCpuReady</i>	2	The CPU is ready.																					
Option	Value	Usage																																
<i>statCpuNotPresent</i>	0	No CPU is present on this port.																																
<i>statCpuNotReady</i>	1	The CPU is not ready.																																
<i>statCpuReady</i>	2	The CPU is ready.																																

Option	Value	Usage
<i>statCpuErrorOsHalt</i>	3	The CPU has encountered an OS error and has halted.
<i>statCpuErrorMemTestFailed</i>	4	The CPU encountered an error during memory tested and has halted.
<i>statCpuErrorBootFailed</i>	5	The CPU failed to completely boot.
<i>statCpuErrorNotResponding</i>	6	The CPU is not responding.

portCpuDodStatus

Read-only. The state of the DOD (software download on demand) process. One of

Option	Value	Usage
<i>statCpuDodNotReady</i>	0	The DOD process has not completed yet.
<i>statCpuDodReady</i>	1	The DOD process has completed.

posK1Byte

Read-only. 64-bit value. The current K1 byte code value being received in the Sonet frame.

posK2Byte

Read-only. 64-bit value. The current K2 byte code value being received in the Sonet frame.

prbsBerRatio

Read-only. 64-bit value. Ratio of PRBS errored bits to bits received.

prbsBitsReceived

Read-only. 64-bit value. Number of PRBS bits received.

prbsErroredBits

Read-only. 64-bit value. Number of PRBS errored bits received.

prbsFramesReceived

Read-only. 64-bit value. Number of PRBS frames received.

prbsHeaderError

Read-only. 64-bit value. Number of PRBS header errors received.

qualityOfService0-7

Read-only. 64-bit value. Number of frames counted by Quality of Service Counter 0 through 7 that meet the criteria set up using the **qos** command. This counter is available when stat mode is set to *statQos*.

remoteFaults

Read-only. 64-bit value. The number of remote link faults detected.

remoteOrderedSetReceived

Read-only. 64-bit value. The number of remote ordered sets received. Ordered sets are part of Link Fault Signaling.

remoteOrderedSetSent

Read-only. 64-bit value. The number of remote ordered sets sent. Ordered sets are part of Link Fault Signaling.

rprDiscoveryFramesReceived

Read-only. 64-bit value. The number of RPR discovery frames received.

rprDataFramesReceived

Read-only. 64-bit value. The number of RPR encapsulated data frames received.

rprFairnessFramesReceived

Read-only. 64-bit value. The number of RPR fairness frames received.

rprFairnessFramesSent

Read-only. 64-bit value. The number of RPR fairness frames sent.

rprFairnessTimeouts	<i>Read-only.</i> 64-bit value. The number of timeouts that occurred waiting for RPR fairness frames.
rprHeaderCrcErrors	<i>Read-only.</i> 64-bit value. The number of RPR frames received with header CRC errors.
rprIdleFramesReceived	<i>Read-only.</i> 64-bit value. The number of RPR idle frames received
rprOamFramesReceived	<i>Read-only.</i> 64-bit value. The number of RPR OAM frames received.
rprPayloadCrcErrors	<i>Read-only.</i> 64-bit value. The number of RPR frames received with payload CRC errors.
rprProtectionFramesReceived	<i>Read-only.</i> 64-bit value. The number of RPR protection frames received.
RRDYsSent	<i>Read-only.</i> Receiver Ready error singal sent.
RRDYsReceived	<i>Read-only.</i> Receiver Ready error singal received.
remoteBBCreditCount	<i>Read-only.</i> The count of the number of remote buffers supported by an FC port.
remoteBBCreditValue	<i>Read-only.</i> The credit value of the remote buffers supported by an FC port.
disparityErrors	<i>Read-only.</i> The error that occurs when hardware wrongly selects 10B code for 8B hex value in the frame. It is 8B10B encoding error and is seen only in 10B encoded data.
nsQuerySent	<i>Read-only.</i> The name server query sent to the FC port.
nsQuerySuccessful	<i>Read-only.</i> The successful transmission of NS Query.
PRLISent	<i>Read-only.</i> The Process Login parameters sent by this port.
PRLIReceived	<i>Read-only.</i> The Process Login parameters received by this port.
PRLISuccessful	<i>Read-only.</i> The Process Login parameters successfully sent and received by this port.
RSCNReceived	<i>Read-only.</i> The Registered State Change Notification (RSCN) ELS received.
RSCNAccTransmitted	<i>Read-only.</i> The Registered State Change Notification (RSCN) ELS transmitted.
SCRTransmitted	<i>Read-only.</i> The State Change Registration (SCR) ELS transmitted.
SCRReceived	<i>Read-only.</i> The State Change Registration (SCR) ELS received.
invalidEOFCount	<i>Read-only.</i> The count of invalid End of Frames.
codeError	<i>Read-only.</i> The error codes sent.
fcFlogiSent	<i>Read-only.</i> The Fabric Login (FLOGI) ELS sent.
fcFlogiLsAccReceived	<i>Read-only.</i> The Link Service Accept (LS_ACC) ELS notification received.

fcPlogiSent	<i>Read-only.</i> The PLOGI ELS notification sent.
fcPlogiLsAccReceived	<i>Read-only.</i> The PLOGI Link Service Accept (LS_ACC) ELS notification received.
fcPlogiRequests Received	<i>Read-only.</i> The PLOGI ELS notification received.
fcFlogoSent	<i>Read-only.</i> The FLOGO notification sent.
fcPlogoSent	<i>Read-only.</i> The PLOGO notification sent.
fcPlogoReceived	<i>Read-only.</i> The PLOGO notification received.
fcFdiscSent	<i>Read-only.</i> The FDISC notification sent.
fcFdiscLsAccReceived	<i>Read-only.</i> The FDISC LS_ACC notification received.
fcNSRegSent	<i>Read-only.</i> The Name Server Registration notification sent.
fcNSRegSuccessful	<i>Read-only.</i> The Name Server Registration notification sent successfully.
fcNxPortsEnabled	<i>Read-only.</i> The Nx port is enabled.
fcNxPortIdsAcquired	<i>Read-only.</i> The ID of Nx port is acquired.
RxPausePriorityGroup 0Frames	<i>Read-only.</i> 64-bit value. The number of Rx Pause Priority Group 0 frames received.
RxPausePriorityGroup 1Frames	<i>Read-only.</i> 64-bit value. The number of Rx Pause Priority Group 1 frames received.
RxPausePriorityGroup 2Frames	<i>Read-only.</i> 64-bit value. The number of Rx Pause Priority Group 2 frames received.
RxPausePriorityGroup 3Frames	<i>Read-only.</i> 64-bit value. The number of Rx Pause Priority Group 3 frames received.
RxPausePriorityGroup 4Frames	<i>Read-only.</i> 64-bit value. The number of Rx Pause Priority Group 4 frames received.
RxPausePriorityGroup 5Frames	<i>Read-only.</i> 64-bit value. The number of Rx Pause Priority Group 5 frames received.
RxPausePriorityGroup 6Frames	<i>Read-only.</i> 64-bit value. The number of Rx Pause Priority Group 6 frames received.
RxPausePriorityGroup 7Frames	<i>Read-only.</i> 64-bit value. The number of Rx Pause Priority Group 7 frames received.
scheduledFramesSent	<i>Read-only.</i> 64-bit value. The number of frames transmitted as part of programmed streams.
scheduledTransmitTime	<i>Read-only.</i> 64-bit value. This only applies to ports that support the portFeatureScheduledTxDuration feature (see port is ValidFeature). This is the

scheduled transmit time associated with the port. This statistic is also available with the *getScheduledTransmitTime* sub-command of this command.

schedulerTemperature

Read-only. 64-bit value. The temperature at the scheduler chip.

sectionBip

Read-only. 64-bit value. Number of section BIP errors received on OC ports for Packet over Sonet interfaces. The *enablePosExtendedStats* options must be *true* for this value to be valid.

sectionBipErroredSecs

Read-only. 64-bit value. A count of the number of seconds during which (at any point during the second) at least one section layer BIP was detected.

sectionLossOfFrame

Read-only. 64-bit value. Number of section LOF indications received on OC ports for Packet over Sonet interfaces. The *enablePosExtendedStats* options must be *true* for this value to be valid.

sectionLossOfSignal

Read-only. 64-bit value. Number of section LOS indications received on OC ports for Packet over Sonet interfaces. The *enablePosExtendedStats* options must be *true* for this value to be valid.

sectionLossOfSignal Secs

Read-only. 64-bit value. A count of the number of seconds during which (at any point during the second) at least one section layer LOS defect was present.

sequenceErrors

Read-only. 64-bit value. Number of sequence errored frames.

sequenceFrames

Read-only. 64-bit value. Number of signature matched frames.

sonetCircuitType

Read-only. The type of the Sonet Circuit. One of:

statSonetCircuitAsymmetric
statSonetCircuitSts1
statSonetCircuitSts3c
statSonetCircuitSts12c
statSonetCircuitSts48c
statSonetCircuitSts_0
statSonetCircuitStm1
statSonetCircuitStm4
statSonetCircuitStm16
statSonetCircuitSts1Xv
statSonetCircuitSts3cXv
statSonetCircuitSts12cXv
statSonetCircuitVc3Xv
statSonetCircuitVc4Xv

sonetTimeslotLcasSink State

Read-only. The state of the Sonet Timeslot LCAS Sink. One of:

statSonetTimeslotSinkStop
statSonetTimeslotSinkIdle
statSonetTimeslotSinkOk
statSonetTimeslotSinkFail

sonetTimeslotLcasSourceState	<i>Read-only.</i> The state of the Sonet Timeslot LCAS Source. One of:
	statSonetTimeslotSourceStop statSonetTimeslotSourceIdle statSonetTimeslotSourceNorm statSonetTimeslotSourceDnu statSonetTimeslotSourceAdd statSonetTimeslotSourceRemove
srpDataFramesReceived	<i>Read-only. 64-bit value.</i> The number of Data frames received. IPv4 frames fall in this category.
srpDiscoveryFramesReceived	<i>Read-only. 64-bit value.</i> The number of topology discovery frames received.
srpIpsFramesReceived	<i>Read-only. 64-bit value.</i> The number of IPS type frames received.
srpParityErrors	<i>Read-only. 64-bit value.</i> The number of SRP frames received with SRP header parity error. This includes all frame types.
srpUsageFramesReceived	<i>Read-only. 64-bit value.</i> The number of usage frames received with good CRC, good header parity and only those that match the MAC address set for the SRP's port. Bad CRC frames, frames with header errors or those with other MAC addresses are received but not counted.
srpUsageFramesSent	<i>Read-only. 64-bit value.</i> The number of usage frames sent. These are sent periodically to keep the link alive.
srpUsageStatus	<i>Read-only. 64-bit value.</i> If the number of consecutive timeouts exceeds the Keep Alive threshold set in srpUsage , this status changes to FAIL. Otherwise shows OK.
srpUsageTimeouts	<i>Read-only. 64-bit value.</i> The number of times a usage frame was not received within the time period set in the srpUsage .
streamTrigger1	<i>Read-only. 64-bit value.</i> User-Defined Statistic counter 5 indicating number of frames received that meet the filtering criteria set up using the filter command. To use this counter the stat mode has to be set to <i>statStreamTrigger</i> .
streamTrigger2	<i>Read-only. 64-bit value.</i> User-Defined Statistic counter 6 indicating number of frames received that meet the filtering criteria set up using the filter command. To use this counter the stat mode has to be set to <i>statStreamTrigger</i> .
symbolErrorFrames	<i>Read-only. 64-bit value.</i> Number of frames received with symbol errors (<i>gigabit only</i>).
symbolErrors	<i>Read-only. 64-bit value.</i> Number of symbol errors.
synchErrorFrames	<i>Read-only. 64-bit value.</i> Number of frames with synchronized errors (<i>gigabit only</i>).
tcpChecksumErrors	<i>Read-only. 64-bit value.</i>
tcpPackets	<i>Read-only. 64-bit value.</i>

tenGigLanRxFpgaTemperature	<i>Read-only.</i> 64-bit value. For 10Gbe: the temperature at the LAN receive FPGA.									
tenGigLanTxFpgaTemperature	<i>Read-only.</i> 64-bit value. For 10Gbe: the temperature at the LAN transmit FPGA.									
transmitDuration	<i>Read-only.</i> 64-bit value. Transmit duration, in nanoseconds.									
transmitState	<i>Read-only.</i> Reflects the current state of transmit. The following states can be read:									
	<table border="1"> <thead> <tr> <th>Option</th> <th>Value</th> <th>Usage</th> </tr> </thead> <tbody> <tr> <td><i>statIdle</i></td> <td>0</td> <td>transmit stopped</td> </tr> <tr> <td><i>statActive</i></td> <td>1</td> <td>port currently transmitting</td> </tr> </tbody> </table>	Option	Value	Usage	<i>statIdle</i>	0	transmit stopped	<i>statActive</i>	1	port currently transmitting
Option	Value	Usage								
<i>statIdle</i>	0	transmit stopped								
<i>statActive</i>	1	port currently transmitting								
udpChecksumErrors	Read-only. 64-bit value.									
udpPackets	Read-only. 64-bit value.									
undersize	<i>Read-only.</i> 64-bit value. Number of undersized frames (less than 64 bytes) received.									
userDefinedStat1	<i>Read-only.</i> 64-bit value. Number of frames counted by User Defined Statistics Counter 1 that meet the criteria set up using the filter command. This counter is available when stat mode is set to <i>statNormal</i> .									
userDefinedStat2	<i>Read-only.</i> 64-bit value. Number of frames counted by User Defined Statistics Counter 2 that meet the criteria set up using the filter command. This counter is available when stat mode is set to <i>statNormal</i> .									
vlanTaggedFramesRx	<i>Read-only.</i> 64-bit value. Number of VLAN Tagged frames received.									
userDefinedStatByteCount1	<i>Read-only.</i> 64-bit value. Number of bytes counted by User Defined Statistics Counter 1 that meet the criteria set up using the filter command. This counter is available when stat mode is set to <i>statNormal</i> .									
userDefinedStatByteCount2	<i>Read-only.</i> 64-bit value. Number of bytes counted by User Defined Statistics Counter 2 that meet the criteria set up using the filter command. This counter is available when stat mode is set to <i>statNormal</i> .									

DEPRECATED OPTIONS

enableUsbExtendedStats true/false	USB support has been removed from IxOS. This option has no effect.
countertype	Deprecated. Use statAllStats.
counterRate	<i>Read-only.</i> 64-bit value. The rate of the value of the statistic counter.
counterVal	<i>Read-only.</i> 64-bit value. The value of the statistic counter.

usbRxBitStuffing
usbRxBufferOverrun
usbRxCRCError
usbRxDataOverrun
usbRxdataUnderrun
usbRxDeviceNot
Responding
usbRxNoError
usbRxNotAccessed
usbRxPIDCheckFail
usbRxStall
usbRxToggleMismatch
usbRxUnexpectedPID
usbTxBufferUnderrun
usbTxDeviceNot
Responding
usbTxNoError
usbTxNotAccessed
usbTxPIDCheckFail
usbTxStallusbTx
UnexpectedPID

USB support has been removed from IxOS. These options maintains a constant value.

COMMANDS

The **stat** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

stat cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **stat** command. Specific errors include:

- Invalid statistic for port.

stat clearBertLane chasID cardID portID

Clears all Bert stats for the port, if the card is 40GE LSM XMV or 100GE LSM XMV.

stat config option value

Modify the configuration options of the statistics. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for capture.

stat get statAllStats chasID cardID portID

Gets the statistics counter for all stats. Note that *counterType* has been deprecated; use **statAllStats** instead. **statAllStats** makes all of the statistics available through the options.

Specific errors are:

- No connection to a chassis
- Invalid port number
- Additional delay is needed between ‘gets’
- Network error between client and chassis

stat getBertChannel *chasID cardID portID level*

For channelized BERT cards, loads the BERT related statistics (with a *bert* prefix) for the *level* indicated. These may then be obtained with normal *stat cget* commands.

stat getBertChannelRate *chasID cardID portID level*

For channelized BERT cards, loads the BERT related rate statistics (with a *bert* prefix) for the *level* indicated. These may then be obtained with normal *stat cget* commands.

stat getBertLane *chasID cardID portID laneNumber*

If the card is 40GE LSM XMV or 100GE LSM XMV, the *laneNumber* option is used to specify the BERT lane.

stat getCaptureState *chasID cardID portID*

Returns the capture state of the port. See the values associated with the **captureState** standard option. Specific errors are:

- No connection to a chassis
- Invalid port number
- Network error between client and chassis

stat getLineSpeed *chasID cardID portID*

Returns the line speed of the port. See the values associated with the **lineSpeed** standard option.

stat getLinkState *chasID cardID portID*

Returns the link state of the port. See the values associated with the **link** standard option.

stat getRate *statAllStats chasID cardID portID*

Gets the frame rate for all stats. Note that *counterType* has been deprecated; use **statAllStats** instead. **statAllStats** makes all of the statistics available through the options.

Specific errors are:

- No connection to a chassis
- Invalid port number
- Network error between client and chassis

stat getScheduledTransmitTime *chasID cardID portID*

Returns the scheduled transmit time of the port. See the values associated with the **scheduledTransmitTime** standard option.

stat getSonetCircuit *chasID cardID portID circuitID*

Gets all circuit-level statistics for circuit with specified ID.

stat getSonetCircuitRate *chasID cardID portID circuitID*

Gets all circuit-level statistics that have rates.

stat getSonetSlot *chasID cardID portID circuitID timeslot timeslotDirection*

Gets all slot-level statistics for slot with specified ID.

stat getSonetSlotRate *chasID cardID portID circuitID timeslot timeslotDirection*

Gets all slot-level statistics that have rates.

stat getTransmitState *chasID cardID portID*

Returns the transmit state of the port. See the values associated with the **transmitState** standard option.

stat getTxIgnoreLinkState *chasID cardID portID*

This command works in conjunction with the *getLinkState* command to determine physical link when the port state *txIgnoreLinkState* is selected on a port.

stat set *chasID cardID portID*

Sets the configuration of the statistics counters on port *portID*, card *cardID*, chassis *chasID* in IxHAL. Note - if the mode is set to anything other than *statNormal*, then up to 8 of the hardware counters are reused for an alternate statistic. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is being used by another user
- The configured parameters are not valid for this port
- Network error between client and chassis

stat setDefault

Sets the stat mode to default and zeros all stat counters.

stat write *chasID cardID portID*

Writes or commits the changes in IxHAL to hardware for port *portID*, card *cardID*, chassis *chasID*. Before using this command, use the *stat set* command to configure the stream related options in IxHAL.

EXAMPLES

```
package require IxTclHal
set host localhost
set username StatExampleUser
ixConnectToChassis $host

# Get the chassis ID to use in port lists.
set chas [ixGetChassisID $host]

# Assume that there's an ethernet card in this slot with proper
# ethernet connections, with port 1 looped to port 2.
set card 1
set portList [list [list $chas $card 1] [list $chas $card 2]]

# Login before taking ownership.
ixLogin $username

if {[ixTakeOwnership $portList]} {
    errorMsg "Error taking ownership"
    return $::TCL_ERROR
```

```

}

# Set factory defaults on all ports in portList.
foreach port $portList {
    scan $port "%d %d %d" chas card port
    if {[setFactoryDefaults $chas $card $port]} {
        ixPuts "Error setting factory defaults on port $chas $card
$port"
        return $::TCL_ERROR
    }
}
# Commit changes to hardware and verify linkState before
continuing.
ixWritePortsToHardware portList
ixCheckLinkState portList

ixClearStats portList
ixStartTransmit portList

# Once per second, get some statistics.
# Note that stats are only polled by hardware every 200-600ms,
depending on the hardware,
# so attempts to retrieve stats more often than 2-3 times per
second will only
# slow down IxServer in an attempt to service the requests.

for {set i 1} {$i <= 5} {incr i} {
    after 1000
    foreach port $portList {
        scan $port "%d %d %d" chas card port
        if {[stat get statAllStats $chas $card $port]} {
            ixPuts "Error reading stats on port $chas $card $port"
            return $::TCL_ERROR
        }
        set framesSent [stat cget -framesSent]
        set framesRecv [stat cget -framesReceived]
        # then a getRate for individual rate stats
        if {[stat getRate statAllStats $chas $card $port]} {
            ixPuts "Error reading stat rate on port $chas $card
$port"
            return $::TCL_ERROR
        }
        set framesSentRate [stat cget -framesSent]
        set framesRecvRate [stat cget -framesReceived]
        ixPuts "Iter $i, Port: $port"
        ixPuts "Frames Sent: $framesSent\trate: $framesSentRate"
        ixPuts "Frames Rcvd: $framesRecv\trate: $framesRecvRate\n"
    }
}
# Also note that the statGroup/statList command pair is not only a
better way to retrieve
# stats on multiple ports, it is the recommended method - see
section <fill in the blank>
# for more details.
ixStopTransmit portList
ixClearOwnership $portList
ixLogout
cleanUp

```

SEE ALSO*statList, statGroup, statWatch*

NAME - statAggregator

statAggregator - gets the aggregated statistics on a list of PGIDs.

SYNOPSIS

statAggregator *sub-command options*

DESCRIPTION

The **statAggregator** command is used to aggregate statistics for a range or list of ranges. In addition, the user selects which packetGroupStats to aggregate as well as the type of aggregation.

STANDARD OPTIONS

packetGroup	Specifies packet group statistics.
totalPGIDs	Specifies statistics on all PGIDs.

COMMANDS

The **statAggregator** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

statAggregator **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **statAggregator** command.

statAggregator **calculate** *option [statList] [fromPgid] [toPgid]*

Computes the aggregate statistics for the selected statistics on the selected range of PGIDs. Enter the option type from a list of options above. Statistics are retrieved and aggregated based on the [statList] options entered.

Statistic	Description
<i>minLatency</i>	Aggregate the minimum latency statistics.
<i>maxLatency</i>	Aggregate the maximum latency statistics
<i>maxminInterval</i>	Aggregate the interval between the minimum and maximum latency statistics
<i>averageLatency</i>	Aggregate the average latency
<i>totalFrames</i>	Aggregate the total number of frames
<i>totalByteCount</i>	Aggregate the total number of bytes
<i>smallSequenceError</i>	Aggregate small sequence errors
<i>bigSequenceError</i>	Aggregate big sequence errors
<i>reverseSequenceError</i>	Aggregate reverse errors
<i>totalSequenceError</i>	Aggregate the total number of sequence errors
<i>sequenceGaps</i>	Aggregate sequence gaps

Statistic	Description
<i>duplicateFrames</i>	Aggregate duplicate frames

What type of aggregation is done is based on an algorithm entered.

Algorithm	Description
<i>avg</i>	Perform an average calculation on the retrieved statistics.
<i>min</i>	Find the minimum value for the retrieved statistics
<i>max</i>	Find the maximum value for the retrieved statistics.
<i>total</i>	Find the total number for the retrieved statistics.

For example, to aggregate the average number of big sequence errors for packet groups 1 to 5, enter:

```
statAggregator calculate packetGroup bigSequenceError avg 1 5
```

More than one statistic and algorithm can be entered per command.

The computation of the aggregated statistics are available until the you either:

- issues a new request for calculate
- calls the *setDefault* method.

Retrieval of new data by the *packetGroupStat get* command will not clear existing aggregated statistics metrics.

statAggregator **setDefault**

Resets the **statAggregator** command to the factory defaults.

EXAMPLES

```
package req IxTclHal

set hostname loopback

if {[ixConnectToChassis $hostname] == ::TCL_ERROR} {
    errorMsg "Error connecting to chassis"
    return 1
}

set chasID [chassis cget -id]
set cardID 1
set portID 1

set fromPg 1
set toPg 200
if {[packetGroupStats get $chasID $cardID $portID $fromPg $toPg]} {
    errorMsg "Error getting packetGroupStats for $chasID / $cardID $portID"
    return
}

# note that these are relative to the get, just like the
```

```
# getGroup command in packetGroupStats.
set range1 {10 100}
set range2 {150 200}
set pgIdRangeList [list $range1 $range2]

set pgIdRangeList {{10 100} {150 200}}
set statList {{minLatency {min max average}} {maxLatency {max}} /
{totalFrames {total}} }

# this does the actual computation on last retrieved stats
statAggregator calculate packetGroup $statList $pgIdRangeList

foreach item [statAggregator cget -packetGroupStats] {
    foreach {statName valueList} $item {
        puts "Stat: $statName"
        foreach value $valueList {
            scan $value "%s %d" algorithm metric
            puts "\t$algorithm: $metric"
        }
    }
}

*** Output will look like this:
Stat: minLatency
    min: 42
Stat: maxLatency
    max: 128
Stat: totalFrames
    sum: 12
    avg: 6
    min: 5
    max: 7
%
```

SEE ALSO

statList, statWatch, stat

NAME - statGroup

statGroup - gets the statistics on a set of ports.

SYNOPSIS	statGroup <i>sub-command options</i>
-----------------	--------------------------------------

DESCRIPTION	The statGroup command is used to create a group of ports for the purpose of retrieving all of the statistics from the group of ports at the same time. Statistics retrieved through the use of the <i>statGroup get</i> sub-command are accessed through the use of the <i>statList</i> command.
--------------------	---

STANDARD OPTIONS

numPorts	<i>Read-only</i> . Indicates the number of ports currently in the list.
-----------------	---

COMMANDS	The statGroup command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

statGroup add *chassisID cardID portID*

Adds the indicated port to the list of ports in the group.

statGroup del *chassisID cardID portID*

Deletes the indicated port from the list of ports in the group.

statGroup cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **statGroup** command.

statGroup get

Gets all of the valid statistics associated with each of the ports in the group. The group is formed by successive calls to *statGroup add*. The values of the statistics are available through the use of the *statList* command.

statGroup setDefault

Resets the list to empty.

EXAMPLES	<pre># add ports to get stats on statGroup setDefault foreach port \$portList { scan \$port "%d %d %d" c l p statGroup add \$c \$l \$p } # get the stats if {[statGroup get]} { ixPuts "Error getting stats for this group" set retCode 1 }</pre>
-----------------	---

```
# read stats
statList setDefault
foreach port $portList {
    scan $port "%d %d %d" c l p
    if {[statList get $c $l $p]} {
        continue
    }
    ixPuts "Frames transmitted: \
            [statList cget -framesSent]"

    if {[statList getRate $c $l $p]} {
        continue
    }
    ixPuts "Transmit rate: [statList cget -framesSent]"
}
```

SEE ALSO

statList, statWatch, stat

NAME - statList

statList - gets the statistics from ports previously collected with *statGroup* or *statWatch*.

SYNOPSIS

statList *sub-command options*

DESCRIPTION

The **statList** command is used to get statistics previously read from the ports using the *statGroup* or *statWatch* command. A single call to *statList get* is used to make all of the **valid** statistics for a port available through subsequent calls to *statList cget*. Similarly, rate statistics are made available through the use of *statList getRate*, followed by calls to *statList cget*.

Note that the *statName* used in *cgets* is formed from the standard option name by prepending *stat* to the name and capitalizing the first letter of the option.
(Example: for the option **framesSent**, the *statName* is **statFramesSent**.)

Refer to the *Ixia Reference Guide* for a list of which statistics are available for particular card modules and under particular circumstances.

STANDARD OPTIONS

See *stat*

The STANDARD OPTIONS associated with *statList* are the same as those associated with *stat*, with the exception of the *enable** and *mode* options.

COMMANDS

The **statList** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

statList **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **stat** command.

statList **get** *chassisID cardID portID*

Makes the statistics associated with a particular port accessible through the use of *statList cget -option*. Refer to the *Ixia Reference Guide* for a list of the statistics names and the cases under which they are available.

statList **getRate** *chassisID cardID portID*

Makes the rate statistics associated with a particular port accessible through the use of *statList cget -option*. Refer to the *Ixia Reference Guide* for a list of the statistics names and the cases under which they are available.

statList **setDefault**

Clears all of the statistics previously collected with *statGroup*.

EXAMPLES

See examples under *statGroup*.

SEE ALSO

statGroup, *statWatch*, *stat*

NAME - statWatch

statWatch - automatically get the statistics on a set of ports.

SYNOPSIS

statWatch *sub-command options*

DESCRIPTION

The **statWatch** command is used to create a group of ports and a list of statistics for the purpose of automatically retrieving all of the statistics in the list from the group of ports at the same time. Statistics are automatically delivered once per second. Statistics are then read using the [statList](#) command.

Multiple stat watches may be created, each with its own ID. Each stat watch contains a list of ports and a list of statistics.

Note that the *statName* used in *addStat* and *delStat* is formed from the standard option name by prepending *stat* to the name and capitalizing the first letter of the option. (For example, for the option **framesSent**, the *statName* is **statFramesSent**.)

Refer to the *Ixia Reference Guide* for a list of which statistics are available for particular card modules and under particular circumstances.

STANDARD OPTIONS

none

COMMANDS

The **statWatch** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

statWatch addPort *watchID chassisID cardID portID*

Adds the indicated port to the list of ports in the stat watch whose ID is *watchID*. Specific errors are:

- The stat watch with ID *watchID* does not exist.
- The port is invalid

statWatch addStat *watchID statName*

Adds the indicated statistic to the list of statistics in the stat watch whose ID is *watchID*. Specific errors are:

- The stat watch with ID *watchID* does not exist.

statWatch addStatRate *watchID statName*

Adds the indicated statistic rate to the list of statistics in the stat watch whose ID is *watchID*. Specific errors are:

- The stat watch with ID *watchID* does not exist.

statWatch create *watchID*

Creates a new stat watch with ID *watchID*. Specific errors are:

- The stat watch with ID *watchID* already exists.

statWatch delPort *watchID chassisID cardID portID*

Deletes the indicated port from the list of ports in the stat watch whose ID is *watchID*. Specific errors are:

- The stat watch with ID *watchID* does not exists.
- The port is invalid

statWatch delStat *watchID statName*

Deletes the indicated statistic from the list of statistics in the stat watch whose ID is *watchID*. Specific errors are:

- The stat watch with ID *watchID* does not exists.
- The *statName* is not in the stat watch port list

statWatch delStatRate *watchID statName*

Deletes the indicated statistic rate from the list of statistics in the stat watch whose ID is *watchID*. Specific errors are:

- The stat watch with ID *watchID* does not exists.
- The *statName* is not in the stat watch port list

statWatch destroy *watchID*

Deletes the stat watch with ID *watchID*. Specific errors are:

- The stat watch with ID *watchID* does not exists.

statWatch start *watchID*

Starts watching the stat watch whose ID is *watchID*. The statistics in the stat watch are regularly delivered for all of the ports in the stat watch. The individual statistics may be read through use of the *statList* command. Specific errors are:

- The stat watch with ID *watchID* does not exists.

statWatch setDefault

Stops and destroys all of the stat watches.

statWatch stop *watchID*

Stops watching the stat watch whose ID is *watchID*. Specific errors are:

- The stat watch with ID *watchID* does not exists.

EXAMPLES

```
set portList { {1 1 1} {1 1 2}}
set statList {statFramesSent statFramesReceived}
set watchID 42

statWatch setDefault

# Create a watch with $watchID
if [statWatch create $watchID] {
    errorMsg "Error creating watch $watchID"
}

# add ports to get stats on
foreach port $portList {
    scan $port "%d %d %d" c l p
```

```

        if [statWatch addPort $watchID $c $l $p] {
            errorMsg "Error adding port $c $l $p to statWatch $watchID"
        }
    }

# Add the stats to the watch
for statItem $statList {
    if [statWatch addStat $watchID $statItem] {
        errorMsg "Error adding $statItem to statWatch $watchID"
    }
    if [statWatch addStatRate $watchID $statItem] {
        errorMsg "Error adding $statItem to statWatch $watchID"
    }
}

# Start the watch with $watchID
if {[statWatch start $watchID]} {
    errorMsg "Error watching stats on statWatch $watchID"
}

# Look at the statistics once per second
for {set i 0} {$i <= 10} {incr i} {
    logMsg "***** Polling $i of 10 *****"
    # Read the stats
    statList setDefault
    foreach port $portList {
        scan $port "%d %d %d" c l p
        logMsg "Port $c $l $p"

        if {[statList get $c $l $p]} {
            continue
        }
        logMsg "\tFrames transmitted: [statList cget -framesSent]"
        logMsg "\tFrames received: \
                           [statList cget -framesReceived]"

        if {[statList getRate $c $l $p]} {
            continue
        }
        logMsg "\tTransmit rate: [statList cget -framesSent]"
        logMsg "\tReceive rate: [statList cget -framesReceived]"
    }
    after 1000
}

# stop the watch
if [statWatch stop $watchID] {
    errorMsg "Error stopping stats on statWatch $watchID"
}

# Destroy the watch
if [statWatch destroy $watchID] {
    errorMsg "Error destroying watch $watchID"
}

```

SEE ALSO*statList, statGroup, stat*

NAME - stream

stream - configure the streams on a port of a card on a chassis.

SYNOPSIS	stream <i>sub-command options</i>
DESCRIPTION	<p>The stream command is used to set up frames and bursts to be transmitted on a port of a card on a chassis. The number of streams that a port supports varies; consult the <i>Ixia Hardware Guide</i> for the exact numbers. A stream consists of bursts of frames separated by inter-frame gap and inter-burst gap (in nanoseconds). The source and destination MAC addresses, number of frames in a stream, pattern type, frame size, and inter-stream gap are some of the parameters that can be specified to shape the desired transmit traffic.</p> <p>For SONET cards which support DCC operation, the optional <i>sequenceType</i> argument used in many of the sub-commands indicates whether the sub-command should apply to flows and/or streams. Flows are used when DCC packets are transmitted at the same time as SPE streams.</p> <p>For ATM cards, is it necessary to set/get stream data to/from a specific queue with the <i>setQueue</i> and <i>getQueue</i> sub-commands. General ATM port options are set using the <i>atmPort</i> command, ATM header options are set using the <i>atmHeader</i> command and the stream queues are managed with the <i>streamQueueList</i> and <i>streamQueue</i> commands. ATM streams may have incrementing and/or random frame sizes, but only 16 of either type. All other streams are forced to fixed size.</p> <p>Some port types support weighted random framesize distributions, as described in <i>weightedRandomFramesize</i>.</p> <p>If a <i>Uniform</i> distribution's minimum value is changed and the new minimum value is not already in one of the distributions, then the distribution is forced to the first random range.</p> <p>If a <i>Uniform</i> distribution's maximum value is changed and the new minimum/maximum values are not used in another distribution, then the distribution is forced to the first random range.</p> <p>The framesize of an ATM packet is set by a combination of the <i>enableCpcsLength</i> and <i>cpcsLength</i> options in this command and the <i>framesize</i> option in the <i>stream</i> command. If <i>enableCpcsLength</i> is set to <i>true</i>, then the ATM frame's size is set from the <i>cpcsLength</i> value only. Otherwise, it is set from the <i>stream</i>'s <i>framesize</i> value and the <i>cpcsLength</i> value is calculated from that.</p> <p>Further, the <i>stream getQueue</i> command resets this command's <i>enableCpcsLength</i> option to <i>false</i>. It is important to correctly set the <i>stream</i>'s <i>framesize</i> value and this command's <i>enableCpcsLength</i> and <i>cpcsLength</i> options after each <i>stream getQueue</i> command and call <i>atmHeader set</i> before the next <i>stream setQueue</i> command.</p>
STANDARD OPTIONS	
asyncIntEnable <i>true/false</i>	When this option is set to <i>false</i> , asynchronous transmit events cannot interrupt the stream. The asynchronous event is logged and is invoked as soon as a synchronous stream permits it. Note that only one asynchronous event of a type

is logged, that is, if the same timer expired twice, only one asynchronous event is logged due to that counter. (*default = false*)

adjustMask

The value to adjust Mask. The options include::

Option
<i>gapNone</i>
<i>gapFrame</i>
<i>gapBurst</i>
<i>gapStream</i>
<i>adjustFrameSizeFixed</i>
<i>adjustFrameSizeMin</i>
<i>adjustFrameSizeMax</i>
<i>adjustNumFrames</i>

bpsRate

If *rateMode* is set to *streamRateModeBps*, then use this value the desired bits per second. (*default = 76190476*)

da

Initial destination MAC address assigned to this stream. Specify this address as six hexadecimal numbers delimited by spaces or colons. For example, the following are valid address formats: { 00 01 02 03 04 05 } and { 00:01:02:03:04:05 }. Note that this option will not update the *isl encapDA* value. (*default = {00 00 00 00 00 00}*)

Note: The MAC address format is very important. A failure to use one of the two designated formats results in incorrect script operation.

daMaskSelect

Selects the bits in the 48-bit destination MAC address that are to be masked by the value set by **daMaskValue**. (*default = {00 00 00 00 00 00}*)

daMaskValue

Value of the masked bits selected by **daMaskSelect** in the destination MAC address. (*default = {00 00 00 00 00 00}*)

daRepeatCounter

Specifies how the destination MAC address is incremented or decremented. Possible values include:

Option	Value	Usage
<i>increment</i>	0	increment the MAC address for as many numDA specified
<i>contIncrement</i>	1	Continuously increment the MAC address for each frame
<i>decrement</i>	2	decrement the MAC address for as many numDA specified
<i>contDecrement</i>	3	Continuously decrement the MAC address for each frame
<i>idle</i>	4	(<i>default</i>) no change to MAC address regardless of numDA
<i>ctrRandom</i>	5	Generate random destination MAC address for each frame
<i>daArp</i>	6	If an ARP response is received, then the first MAC address from the ARP table is used as the DA, else the DA field remains unchanged.

Option	Value	Usage
<i>contJitterTestPattern</i>	7	(For 10GE modules only.) The fixed Continuous Jitter Test Pattern (CJPAT), specified in IETF 802.3ae Annex 48A, is supplied. The data field may not be edited.
<i>contRandomTestPattern</i>	8	(For 10GE modules only.) The fixed Continuous Random Test Pattern (CRPAT), specified in IETF 802.3ae Annex 48A, is supplied. The data field may not be edited.

daStep

If *daRepeatCounter* is set to *increment*, *contIncrement*, *decrement*, or *contDecrement*, and the load module supports an arbitrary step size, then this is the value to increment/decrement the destination address by for each address repetition. (*default = 1*)

dataPattern

Sets up the default data pattern to be inserted into the frames of this stream. *type* may be one of the following values:

Option	Value	Usage
<i>dataPatternRandom</i>	-1	the frame contains random data
<i>allOnes</i>	0	the frame contains all 1's
<i>allZeroes</i>	1	the frame contains all 0's
<i>xAAAAA</i>	2	the frame contains all A's
<i>x5555</i>	3	the frame contains all 5's
<i>x7777</i>	4	the frame contains all 7's
<i>xDDDD</i>	5	the frame contains all D's
<i>xFOFO</i>	6	the frame contains repeating pattern of F0F0's
<i>xOF0F</i>	7	the frame contains repeating pattern of OF0F's
<i>xFF00FF00</i>	8	the frame contains repeating pattern of FF00FF00's
<i>x0FF00FF</i>	9	the frame contains repeating pattern of 0FF00FF's
<i>xFFFFF0000</i>	10	the frame contains repeating pattern of FFFF0000's
<i>x0000FFFF</i>	11	the frame contains repeating pattern of 0000FFFF's
<i>x00010203</i>	12	(<i>default</i>) the frame contains a pattern of incrementing bytes.
<i>x00010002</i>	13	the frame contains a pattern of incrementing 16-bit words.
<i>xFFFEFDFA</i>	14	the frame contains a pattern on decrementing bytes.
<i>xFFFFFFFE</i>	15	the frame contains a pattern of decrementing 16-bit words.
<i>x7E7E7E7E</i>	16	the frame contains a continuous jitter pattern (CJPAT).
<i>x4747476B</i>	17	the frame contains a continuous random pattern (CRPAT).
<i>userpattern</i>	18	select this type to insert user-defined data pattern in the frame

dma

This determines the behaviour of the stream flow. The *mode* may be one of the following:

Option	Value	Usage
<i>contPacket</i>	0	(default) continuously transmit the frames on this stream
<i>contBurst</i>	1	continuously transmit bursts of frames on this stream
<i>stopStream</i>	2	stop all transmission from the port where this stream resides regardless of existence of other streams on this port
<i>advance</i>	3	after all the frames are sent from the current stream, the frames from the next stream on the port are transmitted.
<i>gotoFirst</i>	4	the last stream on the port is set to this mode to begin transmission of frames of the first stream in the list
<i>firstLoopCount</i>	5	the last stream on the port is set to this mode to begin transmission of the first stream in the list for <i>loopCount</i> intervals

enable *true/false*

Enable or disable the stream. If disabled, the frames in this stream will not be transmitted along with the other streams on this port. (*default = true*)

enableDaContinueFromLastValue *true/false*

If *true*, then the MAC Destination Address of the stream will not reset when returning to a stream ID, but continue from the previous stream(*default = false*)

enableIbg *true/false*

Enable the inter-burst gap. (*default = false*)

enableIsg *true/false*

Enable the inter-stream gap. (*default = false*)

enableIncrFrameBurstOverride *true/false*

Enable the packet burst override for increment frame mode. (*default = false*)

enableSaContinueFromLastValue *true/false*

If *true*, then the MAC Source Address of the stream will not reset when returning to a stream ID, but continue from the previous stream(*default = false*)

enableSourceInterface *true/false*

If *true*, then the MAC address and source IP address associated with an interface is used instead of the *sa* value and IP source address. The particular interface to be used is specified in *interfaceDescription*. (*default = false*)

enableStatistic *true/false*

If true, then per-stream transmit statistics are enabled (ATM cards only). (*default = true*)

enableSuspend *true/false*

If true, then stream suspend command is enabled. (*default = false*)

enableTimestamp *true/false*

If *true*, 6 bytes of timestamp are inserted before the CRC of the frame. This was previously known as the *fir* option, which is now deprecated. (*default = false*)

enforceMinGap

When a port which supports this feature is in Advanced Scheduler Mode, then this is the minimum gap that is ever inserted between packets. The smallest value supported is 3. (*default = 12*)

fcs

The FCS error to be inserted in the frame. One of the following:

Option	Value	Usage
<i>streamErrorGood</i>	0	(default) a good FCS to be inserted in the frame
<i>streamErrorAlignment</i>	1	an alignment error to be inserted in the frame (only valid for 10/100)
<i>streamErrorDribble</i>	2	dribble error to be inserted in the frame
<i>streamErrorBadCRC</i>	3	a bad FCS error to be inserted in the frame
<i>streamErrorNoCRC</i>	4	no FCS error to be inserted in the frame

floatRate

Read-only. The *framerate* option expressed as a floating point number.

fpsRate

If *rateMode* is set to *streamRateModeFps*, then use this value the desired frames per second. (default = 148810)

framerate

Read-only. It reflects the actual rate in frames per second that this configured stream transmits at, expressed as a INT.

framesize

Number of bytes in each frame in the stream. All frames in the stream have the same size. See the note in the *DESCRIPTION* section above concerning frame sizes in ATM packets. (default = 64)

frameSizeMAX

The maximum frame size to be used when frame size of type *sizeRandom* is selected. (default = 1518)

frameSizeMIN

The minimum frame size to be used when frame size of type *sizeRandom* is selected. (default = 64)

frameSizeStep

If *frameSizeType* is set to *sizeIncr* and the load module supports an arbitrary step size, then this is the value to increment the frame size by for each repetition. (default = 1)

frameSizeType

May assume one of the following values:

Option	Value	Usage
<i>sizeFixed</i>	0	(default) All frames in the stream where this packet has been defined have a fixed size specified by <i>framesize</i> option
<i>sizeRandom</i>	1	Frames with random sizes are generated on the stream on which the frames are defined. Some ports support weighted random framesize distributions; see <i>weightedRandomFramesize</i> and the note at the beginning of this command.
<i>sizeIncr</i>	2	Every frame generated on the stream has incrementing size.
<i>sizeAuto</i>	3	Frame size is automatically calculated. Used for protocols that have variable frame lengths such as DHCP.

frameType

The type field in the Ethernet frame, which does not apply to the MAC layer frames. (default = {})

gapUnit

Gap may be one of the following unit values:

Option	Value	Usage
gapNanoSeconds	0	(default) Sets units of time for gap to nanoseconds
gapMicroSeconds	1	Sets units of time for gap to microseconds
gapMilliSeconds	2	Sets units of time for gap to milliseconds
gapSeconds	3	Sets units of time for gap to seconds
gapClockTicks	4	Sets units of time for gap to clock ticks of load module card. The number of clock ticks varies between load modules. Ixia recommends that you do not use this option. This option is planned for deprecation in the next release.
gapBytes	4	Sets units of gap in terms of the time needed to transmit a number of bytes.

ibg

Inter-Burst Gap is the delay between bursts of frames in clock ticks (see ifg option for definition of clock ticks). If the IBG is set to 0 then the IBG is equal to the ISG and the IBG becomes disabled. (*default = 960.0*)

ifg

The inter-frame gap specified in clock ticks (*default = 960.0*).

ifgMAX

The maximum inter-frame gap in clock ticks to be used when IFG of type *gapRandom* is selected. (*default = 960.0*)

ifgMIN

The minimum inter-frame gap in clock ticks to be used when IFG of type *gapRandom* is selected. (*default = 960.0*)

ifgType

type may be one of the following values:

Option	Value	Usage
gapFixed	0	(default) the gap between all frames is fixed
gapRandom	1	random size of gap is generated between every frame transmitted (not supported yet)

isg

The inter-stream gap is the delay in clock ticks between stream. This delay comes after the receive trigger is enabled. Setting this option to 0 means no delay. (*default = 960.0*)

loopCount

Number of times to begin transmission of the first stream in the list when **stream config -dma firstLoopCount** is set. (*default = 1*)

name

User specified name of the stream. (*default = ""*)

numBursts

Number of bursts in the stream. If the option *dma* is set to *contBurst* or *contPacket* this option is ignored. (*default = 1*)

numDA

Number of destination MAC addresses the stream is going to be transmitted to. **numDA** must be > 1 to set the **daRepeatCounter** to anything other than idle. (*default = 1*)

numFrames

Number of maximum frames in the stream. If the option *dma* is set to *contPacket* this option is ignored. (*default = 100*)

numSA

Number of source MAC addresses on the stream which is going to transmit frames from. **numSA** must be > 1 to set the **saRepeatCounter** to anything other than idle. (*default = 1*)

packetView

Read-only. Displays the frames as they are going to be transmitted. Note: Shows the first frame when the **transmitMode** is set to portTxPacketStreams and shows all the frames when **transmitMode** is set to portTxPacketFlows. Note that when the *enablePreambleView* option of the *txRxPreamble* command is *true*, then this string includes the preamble's values as the first 8 characters.

pattern

Specify a user-defined pattern of data to be transmitted on this stream. The *dataPattern* option must be set to type *userpattern* or this pattern is ignored. (*default = {00 01 02 03}*)

patternType

Type of given patterns that is inserted in all the frames transmitted on this stream. *type* can be one of the following:

Option	Value	Usage
<i>incrByte</i>	0	(<i>default</i>) increment each byte of the frame during transmission
<i>incrWord</i>	1	increment each word of the frame during transmission
<i>decrByte</i>	2	decrement each byte of the frame during transmission
<i>decrWord</i>	3	decrement each word of the frame during transmission
<i>patternTypeRandom</i>	4	generate random pattern of data during transmission
<i>repeat</i>	5	transmit the same pattern of data in the frame
<i>nonRepeat</i>	6	transmit a fixed pattern of data. Note: <i>Fixed</i> type in IxExplorer.
<i>continuousJitterTestPattern</i>	7	transmit a CJPAT pattern.
<i>continuousRandomTestPattern</i>	8	transmit a CRPAT pattern.

percentPacketRate

If *rateMode* is set to *usePercentRate*, then use this value as a percent of maximum transmit rate for this stream. This command sets all three gaps: IFG, IBG and ISG. For ATM ports, the *streamQueue percentMaxRate* value overrides this value if set after the stream has been configured. (*default = 100.0*)

phyMode

Read-only. For cards which support both Copper and Fiber PHY modes, this command shows the current PHY mode.

Option	Value	Usage
<i>portPhyModeCopper</i>	0	Copper
<i>portPhyModeFiber</i>	1	Fiber

preambleData

(10 Gigabit modules only) The 8 bytes in the preamble of the 10 Gigabit Ethernet frame. For SFD Detect Mode, Bytes 2 through 7 are configurable. For Byte Count Mode, Bytes 2 through 8 are configurable (*default = '55 55 55 55 55 55 55'*).

preambleSize

Number of bytes in the preamble field of the frame. Range is between 2 and 255. (*default = 8*)

priorityGroup

Specifies the priority group of the stream. (*default = 0*) Possible values include:

Option	Value	Usage
<i>priorityGroup0</i>	0	assign priority group 0
<i>priorityGroup1</i>	1	assign priority group 1
<i>priorityGroup2</i>	2	assign priority group 2
<i>priorityGroup3</i>	3	assign priority group 3
<i>priorityGroup4</i>	4	assign priority group 4
<i>priorityGroup5</i>	5	assign priority group 5
<i>priorityGroup6</i>	6	assign priority group 6
<i>priorityGroup7</i>	7	assign priority group 7
<i>priorityGroupControl</i>	15	Does not respond to priority flow control. No incoming priority can be mapped to PFC Queue Control. So the traffic on PFC Queue Control cannot be paused/ flow controlled with priority flow control.

rateMode

Specifies whether to use the ifg or percentPacketRate to calculate stream gap.
Possible values include:

Option	Value	Usage
<i>streamRateModeGap</i> <i>Deprecated: useGap</i>	0	use <i>ifg</i>
<i>streamRateModePercentRate</i> <i>Deprecated: usePercentRate</i>	1	(<i>default</i>) use <i>percentPacketRate</i>
<i>streamRateModeFps</i>	2	use <i>fpsRate</i>
<i>streamRateModeBps</i>	3	use <i>bpsRate</i>

region

Reserved for future use and should always be left at its default value of 0.
(*default = 0*)

returnToId streamID

Configures the stream number (*streamID*) that control loops to. (*default = 1*)

**rxTriggerEnable
*true/false***

When set to *true*, the transmit engine waits for a pulse from the receiver to start the stream. (*default = false*)

sa

Initial source MAC address assigned to this stream. Specify this address as six hexadecimal numbers delimited by spaces or colons. For example, the following are valid address formats: { 00 01 02 03 04 05 } and { 00:01:02:03:04:05 }. Note that this option will not update the *isl encapsSA* value. (*default = {00 00 00 00 00 00}*)

Note: The MAC address format is very important. A failure to use one of the two designated formats results in incorrect script operation.

saMaskSelect

Selects the bits in the 48-bit source MAC address that are to be masked by the value set by **saMaskValue**. (*default = {00 00 00 00 00 00}*)

saMaskValue

Value of the masked bits selected by **saMaskSelect** in the source MAC address. (*default = {00 00 00 00 00 00}*)

saRepeatCounter

Specifies how the source MAC address is incremented or decremented. Possible values include:

Option	Value	Usage
<i>increment</i>	0	increment the MAC address for as many numSA specified
<i>contIncrement</i>	1	continuously increment the MAC address for each frame
<i>decrement</i>	2	decrement the MAC address for as many numSA specified
<i>contDecrement</i>	3	continuously decrement the MAC address for each frame
<i>idle</i>	4	(default) no change to MAC address regardless of numSA
<i>ctrRandom</i>	5	generate random source MAC address for each frame
<i>cpeMacAddress</i>	6	for ports operating in USB mode, use the source MAC address provided by the DUT (customer premise equipment).
<i>contJitterTestPattern</i>	7	generate jitter test pattern
<i>contRandomTest Pattern</i>	8	generate random test pattern

saStep

If *saRepeatCounter* is set to *increment*, *contIncrement*, *decrement*, or *contDecrement*, and the load module supports an arbitrary step size, then this is the value to increment/decrement the destination address by for each address repetition. (default = 1)

sourceInterface Description

If *enableSourceInterface* is *true*, this is the interface's description as set in the *description* option of the *interfaceEntry* command when the interface was defined. (default = "")

startOfDataPattern

(Read-only) Sets the data pattern offset, in bytes.

startOfProtocolPad

(Read-only) Calculates the start offset of protocol pad. To know the starting of protocol pad, first protocolPad option is enabled and then stream is set in stream object.

endOfProtocolPad

(Read-only) Calculates the length of data bytes of protocol pad.

startTxDelay

Displays whether the start delay has been set.

startTxDelayUnit

Displays the unit used in the stream start delay.

Option
<i>startTxDelayNanoSeconds</i> = 0
<i>startTxDelayMicroSeconds</i> = 1
<i>startTxDelayMilliSeconds</i> = 2
<i>startTxDelaySeconds</i> = 3
<i>startTxDelayBytes</i> = 4

**suspendState
*true/false***

(Read-only) When true, the selected stream is suspended.

DEPRECATED OPTIONS**fir true/false**

(*enableTimestamp* should be used instead of this option). If Frame Identity Record (FIR) is set to *true*, 6 bytes of timestamp is inserted before the CRC of the frame.

rateMode

The following *rateMode* options have been deprecated.

Deprecated Option	Value	Usage
<i>useGap</i>	0	use <i>ifg</i>
<i>usePercentRate</i>	1	(<i>default</i>) use <i>percentPacketRate</i>

COMMANDS

The **stream** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

stream adjust chasID cardID portID streamIdList adjustMask

Adjusts the rates and frame sizes on the specified streams (stream id list) without stopping the transmission. The corresponding gaps or frame sizes must be configured prior to calling this method using **stream config** followed by **stream set**.

Note: The *frameSizeType* must be configured and committed prior to adjusting stream frame size.

Note: For a frame size adjustment only the hardware state is modified. The software state is unmodified and is not reflected if you do a chassis refresh. The only way to see the size change is to capture the adjusted stream and view the size of the frames in the capture buffer.

The *adjustMask* options, which can be ORed together, are these:

Option	Value	Usage
<i>gapFrame</i>	2	frame gap
<i>gapBurst</i>	4	burst gap
<i>gapStream</i>	8	stream gap
<i>adjustFrameSizeFixed</i>	32	frame size fixed
<i>adjustFrameSizeMin</i>	64	frame size minimum
<i>adjustFrameSizeMax</i>	128	frame size maximum

stream cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **stream** command.

stream config option value

Modify the configuration options of the stream. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for stream.

stream export *fileName chasID cardID portID [fromStreamID toStreamID sequenceType]*

Exports the current stream contents of the port at *portID*, *cardID*, *chasID* into the file named *fileName*; *fileName* may include a full or relative path. The range of streams is expressed by the range of *fromStreamID* (default = 1) and *toStreamID* (default = 0). If *fromStreamId* is less than or equal to 0 the first stream is used and if *toStreamID* is 0, then all streams are exported. The file produced by this command may be used by the *import* sub-command. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

Specific errors are:

- No connection to a chassis
- Stream IDs are not valid
- Invalid port

stream exportQueue *fileName chasID cardID portID queueID [fromStreamID toStreamID]*

Exports a particular queue numbered *queueId* from the current stream contents of the port at *portID*, *cardID*, *chasID* into the file named *fileName*; *fileName* may include a full or relative path. The range of streams is expressed by the range of *fromStreamID* (default = 1) and *toStreamID* (default = 0). If *fromStreamId* is less than or equal to 0 the first stream is used and if *toStreamID* is 0, then all streams are exported. The file produced by this command may be used by the *importQueue* sub-command. Specific errors are:

- No connection to a chassis
- Queue ID is not valid
- Stream IDs are not valid
- Invalid port

stream get *chasID cardID portID streamID [sequenceType]*

Gets the current configuration of the stream with id *streamID* on port *portID*, card *cardID*, chassis *chasID* from its hardware. Call this command before calling **stream cget option value** to get the value of the configuration option. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

stream getCircuit *chasID cardID portID circuitID streamID [sequenceType]*

For use with ports in VCAT mode only. Gets the current configuration of the stream with id *streamID* in the circuit with *circuitID* on port *portID*, card *cardID*, chassis *chasID* from its hardware. Call this command before calling **stream cget option value** to get the value of the configuration option.

stream getPacketView *packetNum*

Gets the packetView data for a specified *packetNum*. The packetView shows the packets that are about to be transmitted. See *packetView* in the Options section.

stream getQueue *chasID cardID portID queueID streamID*

For use with ATM ports only. Gets the current configuration of the stream with id *streamID* in the queue with *queueID* on port *portID*, card *cardID*, chassis *chasID* from its hardware. Call this command before calling **stream cget option value** to get the value of the configuration option. See the note in the *DESCRIPTION* section above concerning frame sizes in ATM packets.

stream import *fileName chasID cardID portID [sequenceType]*

Imports saved stream contents found in the file *fileName* into the port at *portID*, *cardID*, chassis *chasID*. *fileName* may include a full or relative path. All of the streams found in the file are appended to the currently defined streams. The file used by this command must have been produced by the *export* sub-command. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

Specific errors are:

- No connection to a chassis
- Invalid port
- The card is owned by another user
- *fileName* does not exist
- *fileName* does not contain valid data

stream importQueue *fileName chasID cardID portID queueID*

Imports saved stream contents for a particular queue numbered *queueID* found in the file *fileName* into the port at *portID*, *cardID*, chassis *chasID*. *fileName* may include a full or relative path. All of the streams found in the file are appended to the designated queue in the stream. The file used by this command must have been produced by the *exportQueue* sub-command. Specific errors are:

- No connection to a chassis
- Invalid port
- Invalid Queue ID
- The card is owned by another user
- *fileName* does not exist
- *fileName* does not contain valid data

stream resume *chasID cardID portID streamIdList*

Resume the transmission of the streams specified in *streamIdList*. Packet streams (also known as basic or sequentially scheduled streams) can be suspended and resumed during transmission. When a packet stream is suspended and then

resumed, a persistent UDF continues to count from where it left off when the stream was suspended.

If a currently active stream is suspended, it runs to completion and not execute again until it is resumed.

stream send *chasID cardID portID streamID [sequenceType]*

Send a start transmit on one individual stream: *streamID* to port *portID*, card *cardID*, chassis *chasID*. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Stream send uses the protocol server to send out the stream; therefore, the *-dma* mode setting and *-percentMaxRate* setting is not used. Instead, *contBurst* dma mode is always used with **stream send** command. This command is meant to send out frames at a low rate as a debugging tool.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

stream sendCircuit *chasID cardID portID circuitID streamID*

For use with ports in VCAT mode only. Sends the current configuration of the stream with id *streamID* in the circuit with *circuitID* on port *portID*, card *cardID*, chassis *chasID* from its hardware.

stream sendQueue *chasID cardID portID queueID streamID*

For use with ATM ports only. Sends the current configuration of the stream with id *streamID* in the queue with *queueID* on port *portID*, card *cardID*, chassis *chasID* from its hardware.

See the note in the *DESCRIPTION* section above concerning frame sizes in ATM packets.

stream set *chasID cardID portID streamID [sequenceType]*

Sets the configuration of the stream with id *streamID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL by reading the configuration option values set by the **stream config option value** command. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

stream setCircuit *chasID cardID portID circuitID streamID*

For use with ports in VCAT mode only. Sets the configuration of the stream with id *streamID* on its circuit *circuitID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL by reading the configuration option values set by the **stream config option value** command.

stream setDefault

Sets to IxTclHal default values for all configuration options.

Note: The command **stream setDefault** also overwrites the **udf set** command.

stream setFactoryDefaults *chasID cardID portID streamID [sequenceType]*

Sets factory default values for all configuration options for a particular stream. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

stream setQueue *chasID cardID portID queueID streamID*

For use with ATM ports only. Sets the configuration of the stream with id *streamID* on its queue *queueID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL by reading the configuration option values set by the **stream config option value** command. See the note in the *DESCRIPTION* section above concerning frame sizes in ATM packets.

stream suspend *chasID cardID portID streamIdList*

Suspend the transmission of the specified streams (*streamIdList*). Packet streams (also known as basic or sequentially scheduled streams) can be suspended and resumed during transmission. When a packet stream is suspended and then resumed, a persistent UDF continues to count from where it left off when the stream was suspended.

If a currently active stream is suspended, it runs to completion and not execute again until it is resumed.

stream write *chasID cardID portID streamID [sequenceType]*

Writes or commits the changes in IxHAL to hardware for stream with identification *streamID* on port *portID*, card *cardID*, chassis *chasID*. Before using this command, use the **stream set** command to configure the stream related options in IxHAL. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

stream remove *chasID cardID portID streamID*

Removes a stream with identification *streamID* from port *portID*, card *cardID*, chassis *chasID*.

DEPRECATED COMMANDS

stream **setGaps** *ifg chasID cardID portID streamID [sequenceType]*

A helper command that sets the inter-frame gap, inter-stream gap and inter-burst gap specified by *ifg* gap units for the frames in the stream with id *streamID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL and then commits to hardware. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

stream **setIFG** *ifg chasID cardID portID streamID [sequenceType]*

A helper command that sets the inter-frame gap specified by *ifg* gap units for the frames in the stream with id *streamID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL and then commits to hardware. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

stream **setLoopCount** *loopcount chasID cardID portID streamID [sequenceType]*

A helper command that sets the loopcount in the stream with id *streamID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL and then commits to hardware. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only
<i>streamSequenceTypeFlows</i>	2	apply to flows only

stream **setNumFrames** *numFrames chasID cardID portID streamID [sequenceType]*

A helper or convenience command that sets the number of frames specified by *numFrames* in the stream with id *streamID* on port *portID*, card *cardID*, chassis *chasID* in IxHAL and then commits to hardware. The *sequenceType* optional argument indicates whether the settings apply to all modes or one of the modes.

Option	Value	Usage
<i>streamSequenceTypeAll</i>	0	(default) apply to flows and streams
<i>streamSequenceTypeStreams</i>	1	apply to streams only

Option	Value	Usage
<code>streamSequenceTypeFlows</code>	2	apply to flows only

EXAMPLES

```

package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assume a TXS8 card is in slot 4
set card 4
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Check for missing card
if {[card get $chas $card] != 0} {
    ixPuts "Card $card does not exist"
    exit
}

# In this example, we'll set up two streams on the port:
# Any parameters not mentioned are factory defaults
#
# 1) Name = First
#     Advance to next stream
#     1000 packets per burst
#     10 bursts
#     IPG = 1000ns
#     IBG = 2000ns
#     ISG = 3000ns
#     Data = repeating 55 55

```

```

#      Random frame sizes from 100 - 1000 bytes
#      DA = Arp table
#      SA = 04 05 06 07 08 09 incrementing by 2's
# 2) Name = Last
#      Return to ID # 1 for a count of 10
#      5000 packets per burst
#      1 burst
#      IPG = 10000ns
#      Random frame sizes from 100 - 1000 bytes
#      DA = Arp table
#      SA = 04 05 06 07 08 09

# Then, we'll do another example for an ATM card.
# Two queues are used.
# 1) Queue 1
#      Two streams
#      a) All defaults
#      b) VPI/VCI = 33
# 2) Queue 2
#      One stream
#      All defaults

# Make sure the port is at factory default
port setFactoryDefaults $chas $card $port

# Setup stream 1
stream setDefault
stream config -name          "First"
stream config -dma            advance
stream config -numFrames     1000
stream config -numBursts    10
stream config -gapUnit       gapNanoSeconds
stream config -rateMode      useGap
stream config -ifg           1000
stream config -ifgType       gapFixed
stream config -enableIbg    true
stream config -ibg           2000
stream config -enableIsq    true
stream config -isq           3000
stream config -patternType   repeat
stream config -dataPattern   x5555
stream config -frameSizeType sizeRandom
stream config -frameSizeMIN  100
stream config -frameSizeMAX  1000
stream config -daRepeatCounter daArp
stream config -saRepeatCounter increment
stream config -sa             {04 05 06 07 08 09}
stream config -saStep         2
if [stream set $chas $card $port 1] {
    ixPuts "Can't stream set $chas $card $port 1"
}

# Setup stream 2
stream setDefault
stream config -name          "Last"
stream config -dma            firstLoopCount
stream config -returnToId    1
stream config -loopCount     10
stream config -numFrames     5000
stream config -numBursts    1
stream config -gapUnit       gapNanoSeconds
stream config -rateMode      useGap
stream config -ifg           10000

```

```
stream config -ifgType          gapFixed
stream config -enableIbg        false
stream config -enableIsg        false
stream config -daRepeatCounter daAcp
stream config -saRepeatCounter idle
stream config -sa               {04 05 06 07 08 09}
if [stream set $chas $card $port 2] {
    ixPuts "Can't stream set $chas $card $port 2"
}

ixWritePortsToHardware portList

#####
#
# DCC and SPE flows and streams
#
#####
# Now we'll use an OC192 card with DCC in slot 73
set card 73
set port 1
set portList [list [list $chas $card $port]]

# Check for missing card
if {[card get $chas $card] != 0} {
    ixPuts "Card $card does not exist"
}

# In this example, we'll use an OC192 card with DCC and send
# flows on the DCC and normal streams on the SPE
# We'll set up one stream on each of DCC and SPE
# Any parameters not mentioned are factory defaults
#

# Make sure the port is at factory default and then to correct
DCC/SPE
# Mode
port setFactoryDefaults $chas $card $port
port config -transmitMode           portTxModeDccFlowsSpeStreams
port config -receiveMode            portCapture
if [port set $chas $card $port] {
    ixPuts "Can't port set $chas $card $port"
}

# Setup DCC flow
stream setDefault
stream config -name                "DCC"
stream config -dma                 firstLoopCount
stream config -numFrames          1000
stream config -numBursts          10
stream config -gapUnit             gapNanoSeconds
stream config -rateMode            usePercentRate
stream config -percentPacketRate  100
stream config -ifg                1000
stream config -ifgType            gapFixed
stream config -enableIbg          true
stream config -ibg                2000
stream config -enableIsg          true
stream config -isg                3000
stream config -patternType        repeat
stream config -dataPattern        x5555
stream config -frameSizeType     sizeRandom
stream config -frameSizeMIN      100
stream config -frameSizeMAX      1000
```

```

if [stream set      $chas $card $port 1 streamSequenceTypeFlows] {
    ixPuts "Can't stream set $chas $card $port 1
streamSequenceTypeFlows"
}

# Setup SPE stream
stream setDefault
stream config -name          "SPE"
stream config -dma           firstLoopCount
stream config -returnToId    1
stream config -loopCount     10
stream config -numFrames     5000
stream config -numBursts    1
stream config -gapUnit       gapNanoSeconds
stream config -rateMode      usePercentRate
stream config -percentPacketRate 80
stream config -ifg           10000
stream config -ifgType       gapFixed
stream config -enableIbg    false
stream config -enableIsg    false
if [stream set      $chas $card $port 1 streamSequenceTypeStreams] {
    ixPuts "Can't stream set $chas $card $port 1
streamSequenceTypeStreams"
}

ixWritePortsToHardware portList

#####
#
# ATM port
#####
#
# Assume an ATM card is in slot 74
set card 74
set port 1
set portList [list [list $chas $card $port]]

# Make sure the port is at factory default
port setFactoryDefaults $chas $card $port

# Set up port ATM characteristics
atmPort setDefault
atmPort config -interfaceType      0
atmPort config -enableCoset       false
atmPort config -fillerCell        0
if [atmPort set $chas $card $port] {
    ixPuts "Can't atmPort set $chas $card $port"
}

ixWritePortsToHardware      portList

# Clear out all queues and add queue 1
streamQueueList select $chas $card $port
streamQueueList clear

# Add Queue 1 to port at 100% of line rate
set queueID 1
streamQueueList add
streamQueue    setDefault
streamQueue    config      -rateMode      usePercentRate
streamQueue    config      -percentMaxRate 100.0
if [streamQueue set $chas $card $port $queueID] {

```

```
        ixPuts "Can't streamQueue config $chas $card $port"
    }

    set streamID 1
    # Setup stream 1 - no changes from default
    stream setDefault

    # Use defaults in ATM header
    atmHeader setDefault
    atmHeader config -encapsulation atmEncapsulationLLCBridgedEthernetFCS
    atmHeader config -genericFlowControl 5
    atmHeader config -cellLossPriority 1
    if [atmHeader set $chas $card $port] {
        ixPuts "Can't atmHeader set $chas $card $port"
    }

    # Set queue 1 stream 1
    if [stream setQueue $chas $card $port $queueID $streamID] {
        ixPuts "Can't stream setQueue $chas $card $port $queueID
$streamID"
    }

    # Setup stream 2 in queue 1
    set streamID 2

    stream setDefault

    # change VCI to 33 and set for incrementing 16 times by 1
    atmHeader setDefault
    atmHeader config -vci 33
    if [atmHeader set $chas $card $port] {
        ixPuts "Can't atmHeader set $chas $card $port"
    }
    atmHeaderCounter setDefault
    atmHeaderCounter config -type atmCounter
    atmHeaderCounter config -mode atmIncrement
    atmHeaderCounter config -step 1
    atmHeaderCounter config -repeatCount 16
    if [atmHeaderCounter set $chas $card $port] {
        ixPuts "Can't atmHeaderCounter set $chas $card $port"
    }

    # Set queue 1 stream 2 -
    if [stream setQueue $chas $card $port $queueID $streamID] {
        ixPuts "Can't stream setQueue $chas $card $port $queueID
$streamID"
    }

    # Add Queue 2 to port at 50% of line rate
    set queueID 2
    streamQueueList add
    streamQueue setDefault
    streamQueue config -percentMaxRate 50.0
    if [streamQueue set $chas $card $port $queueID] {
        ixPuts "Can't streamQueue config $chas $card $port"
    }

    # Now one stream in queue 2
    set streamID 1
    stream setDefault

    # Use defaults in ATM header
```

```
atmHeader setDefault
if [atmHeader set $chas $card $port] {
    ixPuts "Can't atmHeader set $chas $card $port"
}
# Set queue 1 stream 1
if [stream setQueue $chas $card $port $queueID $streamID] {
    ixPuts "Can't stream setQueue $chas $card $port $queueID
$streamID"
}

ixWritePortsToHardware portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

port, isl, atmHeader, atmHeaderCounter, atmPort, streamQueue, streamQueueList

NAME - streamExtractorFilter

streamExtractorFilter - configures the stream extraction module's filter properties.

SYNOPSIS

streamExtractorFilter *sub-command options*

DESCRIPTION

The **streamExtractorFilter** command is used to configure the stream extraction module's filter properties on the second and third ports.

STANDARD OPTIONS

destOffset

Sets the offset for the destination address, in bytes.

destOffsetMode

Selects where the offset starts for both the destination address for the filter.

Option	Value	Usage
<i>streamExtractorFilter OffsetStartOfFrame</i>	0	(default) start the offset at the beginning of the packet.
<i>streamExtractorFilter OffsetStartOfIp</i>	1	start the offset at the beginning of the IP header.

destPattern

The destination address pattern to filter for.

enableDest *true / false*

Enables filtering on the destination address. (*default = false*)

**enableSource
*true / false***

Enables filtering on the source address. (*default = false*)

filterType

Selects what address type to filter on.

Option	Value	Usage
<i>streamExtractorMac</i>	0	Filter on MAC address
<i>streamExtractorIpv4</i>	1	Filter on IPv4 address
<i>streamExtractorIpv6</i>	2	Filter on IPv6 address
<i>streamExtractorTcp</i>	3	Filter on TCP address
<i>streamExtractorUdp</i>	4	Filter on UDP address

matchOperation

Selects the type of matching to be performed, either 'and' or 'or.'

Option	Value	Usage
<i>streamExtractorFilterAnd</i>	0	Match both conditions
<i>streamExtractorFilterOr</i>	1	Match either condition

sourceOffset

Sets the offset for the source address, in bytes.

sourceOffsetMode	Selects where the offset starts for both the source address for the filter.									
	<table border="1"> <thead> <tr> <th>Option</th><th>Value</th><th>Usage</th></tr> </thead> <tbody> <tr> <td><i>streamExtractorFilter OffsetStartOfFrame</i></td><td>0</td><td>(default) start the offset at the beginning of the packet.</td></tr> <tr> <td><i>streamExtractorFilter OffsetStartOfIp</i></td><td>1</td><td>start the offset at the beginning of the IP header.</td></tr> </tbody> </table>	Option	Value	Usage	<i>streamExtractorFilter OffsetStartOfFrame</i>	0	(default) start the offset at the beginning of the packet.	<i>streamExtractorFilter OffsetStartOfIp</i>	1	start the offset at the beginning of the IP header.
Option	Value	Usage								
<i>streamExtractorFilter OffsetStartOfFrame</i>	0	(default) start the offset at the beginning of the packet.								
<i>streamExtractorFilter OffsetStartOfIp</i>	1	start the offset at the beginning of the IP header.								
sourcePattern	The source address pattern to filter for.									
COMMANDS	The <i>streamExtractorFilter</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.									
streamExtractorFilter cget option	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the <i>streamExtractorFilter</i> command.									
streamExtractorFilter configure option	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the <i>streamExtractorFilter</i> command.									
streamExtractorFilter get chassisID cardID portID	Gets the current configuration of the <i>streamExtractorFilter</i> for port with id <i>portID</i> on card <i>cardID</i> , chassis <i>chassisID</i> from its hardware. Call this command before calling <i>streamExtractorFilter cget option value</i> to get the value of the configuration option. Specific errors are: <ul style="list-style-type: none"> • No connection to a chassis • The <i>select</i> sub-command has not been called 									
streamExtractorFilter set chassisID cardID portID	Gets the current configuration of the <i>streamExtractorFilter</i> for port with id <i>portID</i> on card <i>cardID</i> , chassis <i>chassisID</i> from its hardware. Call this command before calling									
streamExtractorFilter setDefault	Sets to IxTclHal default values for all configuration options.									
streamExtractorFilter setOffsetDefault	Sets to IxTclHal default values for all configuration options.									

EXAMPLES

```

package req IxTclHal

if {[isUNIX]} {
    if {[ixConnectToTclServer loopback]} {
        ixPuts "Error connecting to Tcl Server loopback "
        return 1
    }
}

```

```

ixConnectToChassis {loopback}

set portList {}

set chassis [chassis cget -id]
set card    35
set port    1

# Filter configuration for inline port: 2
set inlinePortId 2
streamExtractorFilter setDefault
streamExtractorFilter config -enableDest      true
streamExtractorFilter config -enableSource    true
streamExtractorFilter config -destPattern     "22 22 22 22 22 22"
streamExtractorFilter config -sourcePattern   "11 11 11 11 11 11"
streamExtractorFilter config -destOffset      0
streamExtractorFilter config -sourceOffset    6
streamExtractorFilter config -destOffsetMode
streamExtractorFilterOffsetStartOfFrame
streamExtractorFilter config -sourceOffsetMode
streamExtractorFilterOffsetStartOfFrame
streamExtractorFilter config -matchOperation
streamExtractorFilterAnd
streamExtractorFilter config -filterType      streamExtractorMac
streamExtractorFilter set $chassis $card $port $inlinePortId
streamExtractorMacFiltering

streamExtractorModifier setDefault
streamExtractorModifier config -enable        true
streamExtractorModifier config -pattern      "01 02 03 04 05 06"
streamExtractorModifier set $chassis $card $port $inlinePortId
streamExtractorDestMac

streamExtractorFilter setDefault
streamExtractorFilter config -enableDest      true
streamExtractorFilter config -enableSource    true
streamExtractorFilter config -destPattern     "5555:5555:5555:5555:5555:5555"
streamExtractorFilter config -sourcePattern   "66AA:5555:5555:0:5555:5555:14"
streamExtractorFilter config -destOffset      24
streamExtractorFilter config -sourceOffset    30
streamExtractorFilter config -destOffsetMode
streamExtractorFilterOffsetStartOfIp
streamExtractorFilter config -sourceOffsetMode
streamExtractorFilterOffsetStartOfIp
streamExtractorFilter config -matchOperation
streamExtractorFilterAnd
streamExtractorFilter config -filterType      streamExtractorIpV6
streamExtractorFilter set $chassis $card $port $inlinePortId
streamExtractorIpFiltering

streamExtractorFilter setDefault
streamExtractorFilter config -enableDest      true
streamExtractorFilter config -enableSource    true
streamExtractorFilter config -destPattern     42
streamExtractorFilter config -sourcePattern   44
streamExtractorFilter config -destOffset      40
streamExtractorFilter config -sourceOffset    44
streamExtractorFilter config -destOffsetMode
streamExtractorFilterOffsetStartOfIp
streamExtractorFilter config -sourceOffsetMode
streamExtractorFilterOffsetStartOfIp

```

```

streamExtractorFilter config -matchOperation
streamExtractorFilterAnd
streamExtractorFilter config -filterType      streamExtractorUdp
streamExtractorFilter set $chassis $card $port $inlinePortId
streamExtractorProtocolFiltering

# Filter configuration for inline port: 3
set inlinePortId 3
streamExtractorFilter setDefault
streamExtractorFilter config -enableDest      true
streamExtractorFilter config -enableSource    true
streamExtractorFilter config -destPattern     "22 22 22 22 22 22"
streamExtractorFilter config -sourcePattern   "11 11 11 11 11
11"
streamExtractorFilter config -destOffset      0
streamExtractorFilter config -sourceOffset    6
streamExtractorFilter config -destOffsetMode
streamExtractorFilterOffsetStartOfFrame
streamExtractorFilter config -sourceOffsetMode
streamExtractorFilterOffsetStartOfFrame
streamExtractorFilter config -matchOperation
streamExtractorFilterAnd
streamExtractorFilter config -filterType
streamExtractorMac
streamExtractorFilter set $chassis $card $port $inlinePortId
streamExtractorMacFiltering

streamExtractorFilter setDefault
streamExtractorFilter config -enableDest      true
streamExtractorFilter config -enableSource    true
streamExtractorFilter config -destPattern     "5555:5555:5555:5555:5555:555"
streamExtractorFilter config -sourcePattern   "66AA:5555:5555:0:5555:5555:14"
streamExtractorFilter config -destOffset      24
streamExtractorFilter config -sourceOffset    30
streamExtractorFilter config -destOffsetMode
streamExtractorFilterOffsetStartOfIp
streamExtractorFilter config -sourceOffsetMode
streamExtractorFilterOffsetStartOfIp
streamExtractorFilter config -matchOperation
streamExtractorFilterAnd
streamExtractorFilter config -filterType
streamExtractorIpV6
streamExtractorFilter set $chassis $card $port $inlinePortId
streamExtractorIpFiltering

streamExtractorFilter setDefault
streamExtractorFilter config -enableDest      true
streamExtractorFilter config -enableSource    true
streamExtractorFilter config -destPattern     42
streamExtractorFilter config -sourcePattern   44
streamExtractorFilter config -destOffset      40
streamExtractorFilter config -sourceOffset    44
streamExtractorFilter config -destOffsetMode
streamExtractorFilterOffsetStartOfIp
streamExtractorFilter config -sourceOffsetMode
streamExtractorFilterOffsetStartOfIp
streamExtractorFilter config -matchOperation
streamExtractorFilterAnd
streamExtractorFilter config -filterType      streamExtractorUdp
streamExtractorFilter set $chassis $card $port $inlinePortId
streamExtractorProtocolFiltering

```

```
lappend portList [list $chassis $card $port]
ixWritePortsToHardware portList
```

SEE ALSO

streamExtractorModifier

NAME - streamExtractorModifier

streamExtractorModifier - replaces, in real time, UDP Video Client addresses in a monitored flow with the addresses of a monitoring device.

SYNOPSIS

streamExtractorModifier *sub-command options*

DESCRIPTION

The **streamExtractorModifier** command is used to replace, in real time, UDP Video Client addresses in a monitored flow with the addresses of a monitoring device. The packet modification performed (by AFM1000 stream extractor module) allows the video monitor to view many different streams, without ever having to change its addresses.

STANDARD OPTIONS

enable

true / false

This enables packet modification of Dest MAC, IPv4 Dest and UDP Dest ports. (*default = false*)

To enable a modifier, any one of the filters must be enabled (MAC address, IP address, or TCP/UDP).

pattern

If enabled, use this pattern in the outgoing packet instead of the original.

(*default = "00 00 00 00 00 00"*)

COMMANDS

The *streamExtractorModifier* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

streamExtractorModifier **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *streamExtractorModifier* command.

streamExtractorModifier **config** *option*

Modify the configuration options of the *streamExtractorModifier*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *streamExtractorModifier*. Specific errors are:

- No connection to a chassis
- The port is being used by another user

streamExtractorModifier **get** *chassisID cardID portID inlinePortID matcherType*

Gets the pattern matcher settings based on the filter direction (*inlinePortId*) and pattern matcher type from IxTclHal to local IxHal. Specific errors are:

- No connection to a chassis
- The *select* sub-command has not been called

streamExtractorModifier set *chasID cardID portID inlinePortID matcherType*

Sets the pattern matcher settings based on the filter direction (*inlinePortId*) and pattern matcher type from IxTclHal to local IxHal.

Matcher type = the destination address type to be modified.

Option	Value	Usage
streamExtractorDestMac	1	(<i>default</i>) enables the AFM to modify the Destination MAC Address of the packet.
streamExtractorDestIpV4	2	enables the AFM to modify the IPv4 Destination IP Address of the packet
streamExtractorDestUdp	3	enables the AFM to modify the UDP Destination Address of the packet

streamExtractorModifier setDefault

Sets to IxTclHal local defaults.

EXAMPLES

See examples in *streamExtractorFilter*.

SEE ALSO

streamExtractorFilter

NAME - streamQueue

streamQueue - configure an ATM stream queue.

SYNOPSIS	streamQueue <i>sub-command options</i>
DESCRIPTION	The streamQueue command is used to configure the data rate of a stream queue for an ATM port.
STANDARD OPTIONS	
aal5FrameRate	<i>Read-only</i> . The rate for all of the streams in the queue, expressed as an AAL5 frame rate.
aal5PayloadBitRate	<i>Read-only</i> . The rate for all of the streams in the queue, expressed as an AAL5 payload bit rate.
aal5PduBitRate	The rate for all of the streams in the queue, expressed as an AAL5 PDU bit rate. (<i>default = 0.0</i>)
aal5SduBitRate	<i>Read-only</i> . The rate for all of the streams in the queue, expressed as an AAL5 SDU bit rate.
cellBitRate	The rate for all of the streams in the queue, expressed as an cell bit rate. (<i>default = 0.0</i>)
cellRate	<i>Read-only</i> . The rate for all of the streams in the queue, expressed as an cell rate.
enableInterleave <i>true / false</i>	If <i>true</i> , then this particular stream queue's cells may be interleaved with all other stream queues. If <i>false</i> , then all of the cells in the stream queue is transmitted without interleaving from other cells from other stream queues that have this option also set to <i>false</i> . (<i>default = true</i>)
percentMaxRate	Sets the rate of all of the streams in a queue as a percentage of the maximum rate. Any individual stream may set its own rate after this option has been set. This value is automatically changed to reflect the new average transmit rate. (<i>default = 0.0</i>)
rateMode	The means by which the ATM rate is to be set.

Option	Value	Usage
usePercentRate	1	(<i>default</i>) Use the value in <i>percentMaxRate</i> to set the ATM rate.
streamQueueAalPduBitRate	2	Use the value in <i>aal5PduBitRate</i> to set the ATM rate.
streamQueueAalCellBitRate	3	Use the value in <i>cellBitRate</i> to set the ATM rate.

**DEPRECATED
OPTIONS**
aal5BitRate

Read-only. The rate for all of the streams in the queue, expressed as an AAL5 bit rate. Same as *aal5PduBitRate*.

COMMANDS

The *streamQueue* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

streamQueue cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *streamQueue* command.

streamQueue clear chasID cardID portID queueId

Removes all streams from a queue on a port. Specific errors are:

- No connection to a chassis
- Invalid *port* number
- Invalid *queueId* number
- The port is being used by another user

streamQueue config option value

Modify the configuration options of the *streamQueue*. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for *streamQueue*. Specific errors are:

- No connection to a chassis
- Invalid *port* number
- Invalid *queueId* number
- The port is being used by another user

streamQueue get chasID cardID portID queueId

Gets the current configuration of the *streamQueue* for a queue on a port from its hardware. Call this command before calling *streamQueue cget option value* to get the value of the configuration option.

streamQueue set chasID cardID portID queueId

Sets the configuration of the *streamQueue* in IxHAL for a queue on a port by reading the configuration option values set by the **streamQueue config option value** command. Specific errors are:

- No connection to a chassis
- Invalid *port* number
- Invalid *queueId* number
- The port is being used by another user
- Configured parameters are not valid for this setting

streamQueue setDefault

Sets to IxTclHal default values for all configuration options.

A

streamQueue

EXAMPLES

See examples in *stream*.

SEE ALSO

atmHeader, *atmPort*, *stream*, *streamQueueList*

NAME - *streamQueueList*

streamQueueList - manage the stream queues for ATM ports.

SYNOPSIS	<i>streamQueueList sub-command options</i>
DESCRIPTION	The streamQueueList command is used to manage the stream queues associated with ATM ports. The <i>select</i> sub-command must be used to select the port before any of the other sub-commands.
<hr/>	
STANDARD OPTIONS	
averageCellRate	<i>Read-only</i> . The average cell rate across all queues associated with a port, specified in ATM cells per second.
averageDataBitRate	<i>Read-only</i> . The average cell rate across all queues associated with a port, specified in data bits per second.
averageFramerate	<i>Read-only</i> . The average cell rate across all queues associated with a port, specified in frames per second.
averagePercentLoad	<i>Read-only</i> . The average cell rate across all queues associated with a port, specified in a percentage of the maximum rate.
<hr/>	
COMMANDS	The <i>streamQueueList</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
<hr/>	
streamQueueList add	Adds an additional stream queue to the port. A <i>queueId</i> is automatically assigned starting from 0. Specific errors are:
	<ul style="list-style-type: none"> • The <i>select</i> sub-command has not been called • 15 ports already associated with the port
streamQueueList cget option	Returns the current value of the configuration option given by <i>option</i> . <i>Option</i> may have any of the values accepted by the <i>streamQueueList</i> command.
streamQueueList clear	All queues are deleted from the port. Specific errors are:
	<ul style="list-style-type: none"> • The <i>select</i> sub-command has not been called
streamQueueList delete queueId	Deletes a queue from the queue list, where <i>queueId</i> is the queue index - starting at 0. All queues below the deleted queue are renumbered down by one. Specific errors are:
	<ul style="list-style-type: none"> • The <i>select</i> sub-command has not been called • The <i>queueId</i> does not exist.

streamQueueList get *chasID cardID portID*

Gets the current configuration of the *streamQueueList* for port with id *portID* on card *cardID*, chassis *chasID* from its hardware. Call this command before calling *streamQueueList cget option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- The *select* sub-command has not been called

streamQueueList select *chasID cardID portID*

Specifies the port that the other sub-commands and options refers to. Specific errors are:

- No connection to the chassis
- Invalid port specified

streamQueueList setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples in *stream*.

SEE ALSO

atmHeader, *atmPort*, *stream*, *streamQueue*

NAME - streamRegion

streamRegion - manage setting that apply to all streams

SYNOPSIS

streamRegion *sub-command options*

DESCRIPTION

The **streamRegion** command is used to manage several properties that apply to all streams.

STANDARD OPTIONS

gapControlMode

For ports that have the *portFeatureGapControlMode* capability, this controls the manner in which minimum inter-packet gaps are enforced.

Option	Value	Usage
<i>streamGapControlFixed</i>	0	(default) All gaps are a minimum of 12 bytes.
<i>streamGapControlAverage</i>	1	The gaps are averaged to 12 bytes in such a way that the deficit at any point in time is no more than 3 bytes.

totalAverageBpsRate

Read-only. The calculated total average bits per second rate.

totalAverageFpsRate

Read-only. The calculated total average frames per second rate.

totalAveragePercent

Read-only. The calculated total average percent of maximum bit rate.

MaxRate

The *streamRegion* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

streamRegion **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *streamRegion* command.

streamRegion **config** *option value*

Modify the stream region options. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS).

streamRegion **enableGenerateWarningList** *value*

If *true*, enables the validation of all inner stream relationships and generates warnings each time *stream set* is called. See *generateWarningList*, below. If disabled (set to *false*) both the generation of the warning list and the validation for the region are disabled. Additionally, *stream cget -warnings* may not contain the correct warnings for the region. (*default = true*)

streamRegion **generateWarningList** *chasID cardID portID*

Validates all inner stream relationships AND generates the warning list for each stream, returning in a list of lists of strings of warnings per each stream.

Regardless of how *enableGenerateWarningList* is set, this command generates a list of lists of warnings per each stream. If a stream has no warnings, the list is empty.

Note: If warning generation is disabled (by the command *enableGenerateWarningList* = false), you **MUST** call *generateWarningList* before committing to hardware, regardless of whether you care about the warning list or not, because this command validates on all the streams in the region. **Validation is required prior to a write to hardware.**

streamRegion get *chassisID cardID portID*

Gets the current configuration of the *streamRegion* for the indicated port. Call this command before calling *streamRegion cget option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis

streamRegion set *chassisID cardID portID*

Sets the configuration of the stream region for the indicated port.

streamRegion setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

SEE ALSO

stream

NAME - streamTransmitStats

streamTransmitStats - view per-stream transmit statistics

SYNOPSIS

streamTransmitStats *sub-command options*

DESCRIPTION

The **streamTransmitStats** command may be used to retrieve the per-stream transmit statistics. This is automatically enabled for all ports that support this feature; this may be checked through the use of the *port is ValidFeature... portFeaturePerStreamTxStats* command.

Per-stream transmit stats are retrieved by the stream id <number> per configuration on the port. They vary per port per transmit mode. (For example, TXS8 cards are numbered from 1 to 255 for Packet Stream mode, and 1 to 128 for Advanced Scheduler mode. And for ATM cards, statistics can only be displayed for 127 streams.)

Statistics for a block of streams are retrieved through the use of the *get* command. Statistics for disabled streams are set to 0. Statistics for a particular stream are retrieved into the options of this command through the use of the *getGroup* command.

The *getGroup* command uses a ‘1’ based index into the block of streams fetched in the *get* command. For example, if *get* was used to fetch streams 101 through 200, then the statistics for stream 105 may be obtained by calling *getGroup* for index 5.

STANDARD OPTIONS

frameRate

Read-only. 64-bit value. This is the transmit frame rate for the stream, expressed in frames per second. **Note:** this value is calculated on the difference between two successive readings; *streamTransmitStats get* must be called at least twice before valid values are obtained.

A value of 0 is returned for disabled streams.

framesSent

Read-only. 64-bit value. This is the number of frames transmitted. A value of 0 is returned for disabled streams.

numGroups

Read-only. This is the number of stream statistics read by the *get* command.

readTimeStamp

Read-only. Reads the timestamp from when the statistics of a port stream were obtained. .

lastTimeStamp

Read-only. 64-bit value. The last timestamp of the transmitted packet.

Note: Does not apply to LM622MR load module in ATM mode (only).

theoreticalAverageFrameRate

Read-only. Calculates the long-term average frame rate for each stream, based on current stream configuration.

COMMANDS

The **streamTransmitStats** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

streamTransmitStats cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **streamTransmitStats** command.

streamTransmitStats get chasID cardID portID [fromStream] [toStream]

Gets a block of transmit statistics for a range of streams on the indicated port. *fromStream* starts at ‘1’, and *toStream* starts at ‘1’. If *fromStream* is omitted, “1” is used. If both *fromStream* and *toStream* are omitted, only the first stream’s statistics are retrieved.

Statistics can only be collected for the first 127 streams on an ATM port.

Call this command before calling **streamTransmitStats cget option**.

streamTransmitStats getCircuit chasID cardID portID circuitID [fromGroupID] [toGroupID]

Gets a block of transmit statistics for a range of Group IDs on the indicated port and circuit. *fromGroupID* starts at “1”, and *toGroupID* starts at “1”. If *fromGroupID* is omitted, “1” is used. If both *fromGroupID* and *toGroupID* are omitted, only the first group’s statistics is retrieved.

Call this command before calling **streamTransmitStats cget option**.

streamTransmitStats getgroup index

Gets the statistics for a particular stream. *index* is with respect to *fromStream* used in the last call to *get*. That is, if the last call to *get* were:

```
streamTransmitStats get $ch $ca $po 10 20
```

then *index* should be set to 2 if the statistics for stream 11 is required. Call this command before calling **streamTransmitStats cget option**.

streamTransmitStats getQueue chasID cardID portID queueID [fromStream] [toStream]

Gets a block of transmit statistics for a range of streams on the indicated port and queue, for ATM modules. *fromStream* starts at “1”, and *toStream* starts at “1”. If *fromStream* is omitted, “1” is used. If both *fromStream* and *toStream* are omitted, only the first stream’s statistics is retrieved.

Statistics can only be collected for the first 127 streams on an ATM port.

Call this command before calling **streamTransmitStats cget option**.

EXAMPLES

```
package require IxTclHal

set host      woodstock
set retCode   "PASS"

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return "FAIL"
    }
}
```

```

}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return "FAIL"
}

set maxStreams 255

# Get the chassis ID to use in port lists
set chId      [chassis cget -id]
set cardId    1
set portId    3
set portList  [list [list $chId $cardId $portId]]

logMsg "Building streams..."

# Check if the port supports per-stream transmit stats
if {! [port isValidFeature $chId $cardId $portId
portFeaturePerStreamTxStats]} {
    ixPuts "Card $cardId does not support per-stream transmit
stats"
    return "FAIL"
}

# Remove all the stream on the port
port reset $chId $cardId $portId

# Set up test streams
stream setDefault
for {set streamId 1} {$streamId <= $maxStreams} {incr streamId} {
    stream config -name "test stream $streamId"
    if { $streamId < $maxStreams } {
        stream config -dma advance
    } else {
        stream config -dma           firstLoopCount
        stream config -loopCount    10000
    }
    stream config -numFrames     1
    stream set $chId $cardId $portId $streamId
}

ixWriteConfigToHardware portList
ixCheckLinkState portList
ixClearStats portList
ixStartTransmit portList

# Get all of the stream stats
if [streamTransmitStats get $chId $cardId $portId 1 $maxStreams] {
    errorMsg "Error getting streamTransmitStats on port $chId
$cardId $portId"
    return "FAIL"
}

# Get all of the stream stats again for a valid reading
if [streamTransmitStats get $chId $cardId $portId 1 $maxStreams] {
    errorMsg "Error getting streamTransmitStats on port $chId
$cardId $portId"
    return "FAIL"
}

ixPuts "Read [streamTransmitStats cget -numGroups] streams"

```

```
ixPuts "Group\tRate\tFrames Sent"
ixPuts "-----"

# Get data for each stream
for {set streamId 1} {$streamId <= $maxStreams} {incr streamId} {
    if {[streamTransmitStats cget -group $streamId] == ""} {
        errorMsg "Error getting group $streamId on port $chId
$cardId $portId"
        set retCode "FAIL"
        break
    }

    set frameRate [streamTransmitStats cget -frameRate]
    set framesSent [streamTransmitStats cget -framesSent]
    ixPuts "$streamId\t$frameRate\t$framesSent"
}

ixStopTransmit portList
ixClearStats portList

return $retCode
```

SEE ALSO

port, stream

NAME - tableUdf

tableUdf - manage table UDFs.

SYNOPSIS

tableUdf *sub-command options*

DESCRIPTION

The *tableUdf* command is used to define tables of data that is applied at the same time as other UDFs. The *tableUdf* feature is only available for selected ports; the availability of the feature may be tested with the *port is ValidFeature... portFeatureTableUDF* command.

The feature is enabled with the *enable* option. Tables consist of rows and columns. Columns define the locations within a packet that are to be modified, while rows hold the data that is simultaneously applied at the locations indicated by the columns. Columns are defined with *tableUdfColumn*; column attributes include:

- Column name
- Offset and size
- Data format; for example, IPv4 address.

Columns are then added to the table using the *addColumn* sub-command of this command.

Once columns have been defined, data is added to the table, row by row, using the *addRow* sub-command.

Table UDF configurations, including row data, may be saved to disk using the *export* sub-command; a comma separated values (csv) file format is used. Table UDF configurations may be retrieved using the *import* sub-command.

STANDARD OPTIONS

enable *true / false*

Enables the table UDF. (*default = false*)

maxRowSize

Retrieves the maximum size of rows in the table UDF. This command can only be used after the *set* command.

maxNumRows

Retrieves the maximum number of rows in the table UDF. This command can only be used after the *set* command.

numColumns

Read-only. The total number of currently defined columns.

numRows

Read-only. The total number of currently defined rows.

COMMANDS

The *tableUdf* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

tableUdf **addColumn**

Adds a table UDF column as defined in the *tableUdfColumn* command. If a column is added after several other columns have been added and *addRow* has

been called for those columns, default data for the existing number of rows for the new column is filled in for the new column. Specific errors are:

- The options in *tableUdfColumn* are invalid
- The maximum number of columns has been exceeded.

tableUdf addRow *rowValueList*

Adds a row's worth of data to the tableUdf. *rowValueList* must contain an entry for each defined column in the table. Each column must be correctly formatted as per the *formatType* and *customFormat* options of the column in the *tableUdfColumn* command at the time that the column was defined. Specific errors are:

- Incorrect number of list items. The number of list items must be the same as the number of columns.
- Data validation failed for one or more columns.
- No columns have been defined.

tableUdf cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *tableUdf* command.

tableUdf clearColumns

Deletes all of the column definitions and all row data.

tableUdf clearRows

Deletes all of the row data. Column definitions are not affected.

tableUdf config *option value*

Modify the table UDF options. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS).

tableUdf delColumn

Deletes the current column selected through calls to *getFirstColumn*/*getNextColumn*. Specific errors are:

- No currently selected column

tableUdf delRow

Deletes the current row selected through calls to *getFirstRow*/*getNextRow*. Specific errors are:

- No currently selected row.

tableUdf export *filename*

Exports the table UDF configuration to the file indicated by *filename*. Specific errors are:

- Invalid *filename*.

tableUdf get *chasID cardID portID*

Gets the current configuration of the *tableUdf* for port with id *portID* on card *cardID*, chassis *chasID* from its hardware. Note that *stream* *get* must be called

before this sub-command. Call this command before calling *tableUdf* *cget option value* to get the value of the configuration option. Specific errors are:

- No connection to a chassis.
- *stream* *get* has not been called.

tableUdf *getFirstColumn*

Finds the first column in the column list and places the values in the options of the *tableUdfColumn* command. Specific errors are:

- The list is empty.

tableUdf *getFirstRow*

Finds the first row in the table and returns a list with the values from the row. Specific errors are:

- The list is empty.

tableUdf *getNextColumn*

Finds the next column in the column list and places the values in the options of the *tableUdfColumn* command. *getFirstColumn* must have been called before this call. Specific errors are:

- No more columns in the list.

tableUdf *getNextRow*

Finds the next row in the table and returns a list with the values from the row. *getFirstRow* must have been called before this call. Specific errors are:

- No more rows in the list.

tableUdf *import* *filename* [*chasID cardID portID*]

Imports the table UDF configuration from the file indicated by *filename*. If *chasID*, *cardID* and *portID* are provided, then this sub-command performs a *tableUdf set* operation as well, committing the values to the hardware. Specific errors are:

- Invalid *filename*.

tableUdf *reserveRows* *numberOfRows*

Reserves a number of rows. This may improve performance by reserving some memory ahead of time so that the process of adding the rows can run faster.

tableUdf *set* *chasID cardID portID*

Sets the configuration of the *tableUdf* in IxHAL for a port by reading the configuration option values set by the *tableUdf config option value* command. Specific errors are:

- No connection to a chassis
- Invalid port specification
- Table UDFs are not supported on this port.
- The port is being used by another user

tableUdf setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package req IxTclHal

set hostname loopback
set retCode "PASS"

if { [ixConnectToChassis $hostname] } {
    errorMsg "error connecting $hostname chassis"
    return "FAIL"
}

set chassisId      [chassis cget -id]
set cardId         56
set portId         1
set customFormat   "8b;3d;16x"

set columnItemList { \
    {"Hex Value"      0     8  formatTypeHex      }  \
    {"Ascii"          20    9  formatTypeAscii} \
    {"Mac Address"   40     6  formatTypeMAC     }  \
    {"Binary Value"  55     2  formatTypeBinary} \
    {"IPV4 Address" 60     4  formatTypeIPv4    }  \
    {"Ipv6 Address" 70    16  formatTypeIPv6    }  \
    {"Decimal"        90    3  formatTypeDecimal} \
    {"Custom Field" 100    4  formatTypeCustom } }

set rowValueListArray(1) {{21 11 11 11 11 11 11 11 12 } hellooooo
{12 12 12 12 12 12 } {00000011 11111111} 1.1.1.2
3A37:3737:373A:3939:3939:3A39:3939:3900 1234 {10000001;4;13DA} }
set rowValueListArray(2) {{31 11 19 99 99 05 00 02 } {arev dzez}
{13 13 13 13 13 13 } {01111111 11111111} 1.1.1.3
3A36:3746:463A:4645:3333:3A31:3233:3400 1235 {10000011;5;CFDF}}
set rowValueListArray(3) {{14 14 14 14 15 15 15 15 } {tgha ari}
{00 14 14 14 14 14 } {00000000 10101010} 1.1.1.4
3A36:3746:463A:4645:3333:3A32:3334:3500 1238 {10000111;6;ABCD}}
set rowValueListArray(4) {{0A CF DB AB AB AB 00 04 } ...mer})_+
{00 15 15 15 15 15 } {00011111 11110001} 1.1.1.5
3A36:3746:463A:4645:3333:3A33:3435:3600 1237 {01001111;7;00AB}}
set rowValueListArray(5) {{21 77 77 77 77 77 77 78 } ...kyank=
{00 00 16 16 16 16 } {00111111 00001111} 1.1.1.6
3A36:3746:463A:4645:3333:3A31:3233:3400 1239 {00000000;0;0AAA} }

set portList [list [list $chassisId $cardId $portId]]

set numColumns [llength $columnItemList]

set columnIndex 1
tableUdf setDefault
tableUdf clearColumns
tableUdf config -enable $::true

foreach formatItem $columnItemList {
    tableUdfColumn      setDefault
```

```

tableUdfColumn config -name      [lindex $formatItem 0]
tableUdfColumn config -offset     [lindex $formatItem 1]
tableUdfColumn config -size       [lindex $formatItem 2]
tableUdfColumn config -formatType [lindex $formatItem 3]
tableUdfColumn config -customFormat $customFormat

if {[tableUdfaddColumn]} {
    errorMsg "Error adding a column with formatType: \
              [lindex $headerItem 3] : $::ixErrorInfo"
    set retCode "FAIL"
    break
}
incr columnIndex
}

set rowIndex 1
foreach rowItem [array names rowValueListArray] {
    if {[tableUdf addRow $rowValueListArray($rowItem)]} {
        errorMsg "Error adding row $rowIndex : $::ixErrorInfo"
        set retCode "FAIL"
        break
    }
}

if { $retCode == "FAIL" } {
    return $retCode
}

if {[tableUdf set $chassId $cardId $portId]} {
    errorMsg "Error setting tableUdf: $::ixErrorInfo"
    return "FAIL"
}

stream setDefault
stream config -name           "tableUdfTester"
stream config -framesize      300

if [stream set $chassId $cardId $portId 1] {
    errorMsg "Error setting stream on port \
              $chassId $cardId $portId 1"
    return "FAIL"
}

if [stream get $chassId $cardId $portId 1] {
    errorMsg "Error getting stream on port $chassId $cardId \
              $portId 1"
    set retCode "FAIL"
    break
}

if [tableUdf get $chassId $cardId $portId] {
    errorMsg "Error getting tableUdf: $::ixErrorInfo"
    return "FAIL"
}

if {[tableUdf cget -enable]} {

    if {[tableUdf cget -enable]} {
        ixPuts "tableUdf cget -enable: [tableUdf cget -enable]"
        set columnIndex 1
        if {![tableUdf getFirstColumn]} {
            ixPuts "***** Column $columnIndex *****"
        }
    }
}

```

```

set fType [tableUdfColumn cget -formatType]
ixPuts "tableUdfColumn cget -formatType: $fType"
if {$fType == ::formatTypeCustom} {
    ixPuts "tableUdfColumn cget -customFormat: \
            [tableUdfColumn cget -customFormat]"
}
ixPuts "tableUdfColumn cget -name: \
        [tableUdfColumn cget -name]"
ixPuts "tableUdfColumn cget -offset: \
        [tableUdfColumn cget -offset]"
ixPuts "tableUdfColumn cget -size: \
        [tableUdfColumn cget -size]"

while {! [tableUdf getNextColumn]} {
    incr columnIndex
    ixPuts "***** Column $columnIndex *****"
    set fType [tableUdfColumn cget -formatType]
    ixPuts "tableUdfColumn cget -formatType: $fType"
    if {$fType == ::formatTypeCustom} {
        ixPuts "tableUdfColumn cget -customFormat: \
                [tableUdfColumn cget -customFormat]"
    }
    ixPuts "tableUdfColumn cget -name: \
            [tableUdfColumn cget -name]"
    ixPuts "tableUdfColumn cget -offset: \
            [tableUdfColumn cget -offset]"
    ixPuts "tableUdfColumn cget -size: \
            [tableUdfColumn cget -size]"
}

set rowIndex 1
set numRows [tableUdf cget -numRows]
if {$numRows > 0 } {
    set rowValueList [tableUdf getFirstRow]
    while {[llength $rowValueList]} {
        ixPuts "***** Row $rowIndex *****"
        ixPuts "$rowValueList"
        set rowValueList [tableUdf getNextRow]
        incr rowIndex
    }
}
}
}

```

SEE ALSO

udf, tableUdfColumn, stream

NAME - tableUdfColumn

tableUdfColumn - manage a table UDF column.

SYNOPSIS

tableUdfColumn *sub-command options*

DESCRIPTION

The *tableUdfCommand* command is used columns used in table UDFs. Columns define the locations within a packet that are to be modified. Columns are defined with the options of this command and then added to a table using the *addColumn* sub-command of the *tableUdf* command. Column attributes include:

- Column name
- Offset and size—data for multiple columns may not overlap
- Data format; for example, IPv4 address.

Column data for existing tables is retrieved with the *getFirstColumn* and *getNextColumn* sub-commands of the *tableUdf*, the values retrieved are available in this command.

Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in *atmHeader*. The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command is with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

STANDARD OPTIONS

customFormat

If *formatType* is set to *formatTypeCustom*, then this string indicates the type of formatting expected. A custom format consists of any number of fixed width fields. Each field has a specific format and fields are separated by one of a number of pre-defined separators. For example: *8b;3d;16x* is a custom format that requires 8 binary digits, a semi-colon, 3 decimal digits, a semi-colon and 16 hex digits. The possible format characters are:

Option	Usage
<i>a</i>	Ascii characters, optionally surrounded by quotes.
<i>b</i>	Binary characters (0 or 1).
<i>d</i>	Decimal characters (0 through 9).
<i>x</i>	Hex characters (0 through 9, a through f, or A through F).

The legal separators are ‘.’, ‘;’, ‘,’, ‘/’ and *space*. (*default* = ““)

formatType

The expected format of the data in the column. Data is expected and is displayed in this format.

Option	Value	Usage
<i>formatTypeHex</i>	0	(<i>default</i>) Hex digits, without any leading ‘0x’ or ‘0X’.
<i>formatTypeAscii</i>	1	Ascii characters. If a space is part of the string, the entire string should be enclosed in quotes.

Option	Value	Usage
<i>formatTypeBinary</i>	2	Binary characters, without any leading '0b' or '0B'.
<i>formatTypeDecimal</i>	3	Decimal characters.
<i>formatTypeMAC</i>	4	A MAC address: 12 hex digits, with or without spaces. If spaces are used, the entire address should be enclosed in quotes.
<i>formatTypeIPv4</i>	5	An IPv4 IP address: four decimal octets separated by periods ('.') .
<i>formatTypeIPv6</i>	6	An IPv6 address.
<i>formatTypeCustom</i>	7	A custom specification, as detailed in <i>customFormat</i> .

name

The name of the column. (*default* = "New Field")

offset

The offset, in bytes, from the beginning of the packet to the start of the column's data. (*default* = 0)

size

The size, in bytes, of the column's data. (*default* = 4)

COMMANDS

The *tableUdfColumn* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

tableUdfColumn cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *tableUdfColumn* command.

tableUdfColumn config *option value*

Modify the table UDF column options. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS).

tableUdfColumn setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

See examples under *tableUdf*.

SEE ALSO

udf, tableUdf, stream

NAME - TCLTransceiver

TCLTransceiver - configure the TCLTransceiver parameters for transceivers.

SYNOPSIS

TCLTransceiver *sub-command options*

DESCRIPTION

The **TCLTransceiver** command is used to configure the TCLTransceiver-specific information of the SFPPlus, CfpQsfp, QSFP, CfpCxp,Cxp, and FCPhy transceivers.

STANDARD OPTIONS

enableMonitorLosProperty	Indicates whether the <i>MonitorLosProperty</i> is enabled.
enableMonitorModuleReadySignalProperty	Indicates whether the <i>MonitorModuleReadySignalProperty</i> is enabled.
enableAutomaticDetectProperty	Indicates whether the <i>AutomaticDetectProperty</i> is enabled.
typeProperty	Indicates the type of transceiver. Possible values are the following:

Option	Values	Usage
<i>kLimiting</i>	0	
<i>kLinear</i>	1	
<i>kTwinax</i>	2	<i>Active copper</i>
<i>kPassiveCopper</i>	3	<i>Passive copper</i>
<i>kNotDetected</i>	4	
<i>kInvalid</i>	5	

When set, you need to use the number corresponding to the desired option.

lockedtypeProperty

Indicates the *lockedtype* property. Possible values are the following:

Option	Values	Usage
<i>kLimiting</i>	0	
<i>kLinear</i>	1	
<i>kTwinax</i>	2	<i>Active copper</i>
<i>kPassiveCopper</i>	3	<i>Passive copper</i>
<i>kNotDetected</i>	4	
<i>kInvalid</i>	5	

When set, you need to use the number corresponding to the desired option.

laneSelectionProperty

Indicates the selection of available lanes. Possible values are the following:

Option	Values
<i>kAllLane</i>	0
<i>kLane1</i>	
<i>kLane2</i>	
<i>kLane3</i>	
<i>kLane4</i>	
<i>kLane5</i>	
<i>kLane6</i>	
<i>kLane7</i>	
<i>kLane8</i>	
<i>kLane9</i>	
<i>kLane10</i>	

Currently, only 4 and 10 lines are supported. For 4 lines, the valid range is from the following:

Option	Values
<i>kAllLane</i>	0
<i>kLane1</i>	
<i>kLane2</i>	
<i>kLane3</i>	
<i>kLane4</i>	

When set, you can use both the number corresponding to the option and the option name.

laneCountProperty

Signifies the number of lanes available for a specific transceiver. Currently only 4 and 10 lines are supported.

txMainTapControlValue Property

Represents the current set value for *txMainTapControl*. Valid range is between *txMainTapControlValueMinProperty* and *txMainTapControlValueMaxProperty*.

txMainTapControlValue MinProperty

Represents the minimum value for *txMainTapControl* property.

txMainTapControlValue MaxProperty

Represents the maximum value for *txMainTapControl* property.

txMainTapControlValue DefaultProperty

Represents the default recommended value for *txMainTapControl*.

txMainTapControlValue PassiveCuLengthProperty

Represents default recommended values for 1, 3, 5 meter copper for *txMainTapControl*.

When reading values from a transceiver that supports multiple lines the value will be in the following format:

	:val1:val2:val3:val4...:valN, where N can be 4 or 10. These are the values for the same property but on different lines(line1 to lineN). Applies only for current values.
txPreTapControlValueProperty	Represents the current set value for <i>txPreTapControlValue</i> . Valid range is between <i>txPreTapControlValueMinProperty</i> and <i>txPreTapControlValueMaxProperty</i> .
txPreTapControlValueMinProperty	Represents the minimum value for <i>txPreTapControlValue</i> property.
txPreTapControlValueMaxProperty	Represents the maximum value for <i>txPreTapControlValue</i> property.
txPreTapControlValueDefaultProperty	The default recommended value for <i>txPreTapControlValue</i> .
txPreTapControlValuePassiveCuLengthProperty	The default recommended values for 1, 3, 5 meter copper for <i>txPreTapControlValue</i> . When reading values from a transceiver that supports multiple lines the value will be in the following format: :val1:val2:val3:val4...:valN, where N can be 4 or 10. These are the values for the same property but on different lines (line1 to lineN). Applies only for current values.
txPostTapControlValueProperty	Represents the current set value for <i>txPostTapControlValue</i> . Valid range is between <i>txPostTapControlValueMinProperty</i> and <i>txPostTapControlValueMaxProperty</i> .
txPostTapControlValueMinProperty	Represents the minimum value for <i>txPostTapControlValue</i> .
txPostTapControlValueMaxProperty	Represents the maximum value for <i>txPostTapControlValue</i> .
maximum value for txPostTapControlValue	The default recommended value for <i>txPostTapControlValue</i> .
txPostTapControlValuePassiveCuLengthProperty	The default recommended values for 1, 3, 5 meter copper for <i>txPostTapControlValue</i> . When reading values from a transceiver that supports multiple lines the value will be in the following format: :val1:val2:val3:val4...:valN, where N can be 4 or 10. These are the values for the same property but on different lines(line1 to lineN). Applies only for current values.
txSlewRateProperty	Represents the current set value for <i>txSlewRate</i> . Valid range is between <i>txSlewRateMinProperty</i> and <i>txSlewRateMaxProperty</i> .
txSlewRateMinProperty	Represents the minimum value for <i>txSlewRate</i> .
txSlewRateMaxProperty	Represents the maximum value for <i>txSlewRate</i> .

txSlewRateDefaultProperty	Represents the default recommended value for <i>txSlewRate</i> .
txSlewRateTapControlValuePassiveCuLengthProperty	Represents the default recommended values for 1, 3, 5 meter copper for <i>txSlewRate</i> . When reading values from a transceiver that supports multiple lines, the value will be in the following format: :val1:val2:val3:val4...:valN, where N can be 4 or 10. These are the values for the same property but on different lines (line1 to lineN). Applies only for current values.
txRiseFallTimeProperty	Represents the current set value for <i>txRiseFallTime</i> . Valid range is between <i>txRiseFallTimeMinProperty</i> and <i>txRiseFallTimeMaxProperty</i> .
txRiseFallTimeMinProperty	Represents the minimum value for <i>txRiseFallTime</i> .
txRiseFallTimeMaxProperty	Represents the maximum value for <i>txRiseFallTime</i> .
txRiseFallTimeDefaultProperty	Represents the default recommended value for <i>txRiseFallTime</i> .
txRiseFallTimeTapControlValuePassiveCuLengthProperty	Represents the default recommended values for 1, 3, 5 meter copper for <i>txRiseFallTime</i> . When reading values from a transceiver that supports multiple lines, the value will be in the following format: :val1:val2:val3:val4...:valN, where N can be 4 or 10. These are the values for the same property but on different lines (line1 to lineN). Applies only for current values.
rxEqualizerControlValueProperty	Represents the current set value for <i>rxEqualizerControlValue</i> . Valid range is between <i>rxEqualizerControlValueMinProperty</i> and <i>rxEqualizerControlValueMaxProperty</i> .
rxEqualizerControlValueMinProperty	Represents the minimum value for <i>rxEqualizerControlValue</i> .
rxEqualizerControlValueMaxProperty	Represents the maximum value for <i>rxEqualizerControlValue</i> .
rxEqualizerControlValueDefaultProperty	Represents the default recommended value for <i>rxEqualizerControlValue</i> .
rxEqualizerTapControlValuePassiveCuLengthProperty	Represents the default recommended values for 1, 3, 5 meter copper for <i>rxEqualizerControlValue</i> . When reading values from a transceiver that supports multiple lines, the value will be in the following format: :val1:val2:val3:val4...:valN, where N can be 4 or 10. These are the values for the same property but on different lines (line1 to lineN). Applies only for current values.

carrierPowerOnProperty
 Y

This is a boolean, should be 1 or 0.

loopbackModeProperty

Valid values are one of the following:

Option	Values
<i>cpfQsfpLoopbackNormal</i>	0
<i>cpfQsfpInternalLoopback</i>	1
<i>cpfQsfpLineLoopback</i>	2
<i>kLane3</i>	
<i>kLane4</i>	

When set, either value or the option name can be used.

manufacturerProperty

Indicates a string representing the manufacturer name.

modelProperty

Indicates a string representing the model name.

laserOnProperty

A boolean value; should be 1 or 0

COMMANDS

The **TCLTransceiver** command is invoked with the following sub-commands.

cget

This command is automatically generated by TCL parser. Since there are no public members to read directly from transceiver object, this command is not used to read properties.

configure

This command is automatically generated by TCL parser. Because there are no public members to configure directly from transceiver object, this command is not used to write properties.

setDefault

Sets a specific transceiver from a specific port in the default state. It is used for initializing the local TCL transceiver object; this default state is only local and the default values set with this command are not necessarily the same as the ones recovered from default properties (like *txMainTapControlValueDefaultProperty*, *txPreTapControlValueDefaultProperty*, *txPostTapControlValueDefaultProperty*, *rxEqualizerControlValueDefaultProperty*).

set

Moves the information set in the transceiver (with *setDefault* or *setValue*) from local TCLHal object to Hal object, just before calling a *write* command (*writePluginToServer*, *saveCustomSetting*).

get

Gets whatever information was received in the transceiver from IxServer, from loGal Hal object to local TCIHal object, just after calling a read command

(*getCurrentSettings*, *getCustomSettings*); in case this command is forgotten, the property's value will not be changed, when using *getValue*.

getReadAvailableProps

Returns a list of available properties which can be read for a specific transceiver on a specific port.

getWriteAvailableProps

Returns a list of available properties which can be written for a specific transceiver on a specific port.

getValue

Returns the current value from TCLHal object for the specific property.

setValue

Sets the specified value for the specified property.

getTransceiverType

Returns the transceiver type or un unsupported type in case neither transceiver type is supported by that port.

Possible values:

- cfpQsfpType
- cfpCxpType
- cxpType
- qsfpType
- sfpPlusType
- fcPhyType
- unsupportedType
- cfp4Qsfp28Type

writePluginToServer

Commits whatever information was set in the Hal object to IxServer. In order to save those properties values, you need to call *setValue*, *set* followed by current command.

saveCustomSetting

Triggers a save custom default command which will save whatever information related to TAP is in the Hal object, into an xml at IxServer level. In order to save those properties values, you need to call *setValue*, than *set* followed by current command.

saveCustomSetting

Triggers a save custom default command which will save whatever information related to TAP is in the Hal object, into an xml at IxServer level. In order to save the values of those properties, you need to call *setValue*, then *set* followed by current command.

applyDefaultSetting

This is similar in functionality with the *applyDefaultSettings* button from IxExplorer. This sets the Tap properties to the values which those properties had taken before the *Ok* button was clicked.

applyCustomSetting

Apply the custom settings (if existing) to the current settings; same behavior as the applyCustomSettings from IxExplorer.

findCustomSetting

Returns if customSettings are available for a specific command (deleteCustom, getCustom, applyCustom, saveCustom).

deleteCustomSetting

Removes the existing custom settings from the xml in IxServer; similar behavior as the deleteCustomSettings button.

getCurrentSettings

Updates the local hal with a copy of current tap values from IxServer. In order to check the values of these properties, you need to run the get command followed by getValue.

getCustomSettings

Updates the local hal with a copy of custom tap values from IxServer. In order to check those properties values you need to run the get command followed by getValue.

getResponseResult

Return the response to findCustomSetting. Depending on the signature parameter of findCustomSetting, its values can be :

```
kDefaultResult = 0,  
kTCLApplyCustomSettingSucceeded = 1,  
kTCLApplyCustomSettingFailed = 2,  
kTCLSaveCustomSettingSucceeded =3,  
kTCLSaveCustomSettingFailed= 4 ,  
kTCLDeleteCustomSettingSucceeded=5,  
kTCLDeleteCustomSettingFailed=6,  
kTCLGetCustomSettingsSucceeded=7,  
kTCLGetCustomSettingsFailed=8,  
kTCLGetCurrentSettingsSucceeded=9,  
kTCLGetCurrentSettingsFailed=10
```

EXAMPLES

```
package require IxTclHal  
  
ixConnectToChassis localhost  
set chassisId [chassis cget -id]  
set cardId 171  
set portId 1  
  
transceiver setDefault $chassisId $cardId $portId  
transceiver getCurrentSettings $chassisId $cardId $portId  
transceiver get $chassisId $cardId $portId
```

```
set propList [transceiver getReadAvailableProps $chassisId $cardID  
$portID]  
foreach element $propList {  
    puts $element  
    puts [transceiver getValue $element]  
}
```

SEE ALSO

stream, protocol, ip

NAME - `tcp`

`tcp` - configure the TCP parameters for a port on a card on a chassis.

SYNOPSIS	<code>tcp sub-command options</code>
DESCRIPTION	<p>The <code>tcp</code> command is used to configure the TCP-specific information used when building TCP type packets if <code>ip config -ipProtocol</code> has been set to <code>Tcp</code>. See RFC 793 for a complete definition of TCP header fields. Note that <code>stream</code> get must be called before this command's get sub-command.</p> <p>Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in <code>atmHeader</code>. The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command are with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.</p>
<hr/>	
STANDARD OPTIONS	
acknowledgement Number	Next byte that the receiver expects from the sending host. (<i>default = 0</i>)
acknowledgeValid <i>true/false</i>	Indicates whether the acknowledgement number field is valid. (<i>default = false</i>)
checksum	If <code>useValidChecksum</code> is set to <code>valid</code> or <code>invalid</code> , this is the TCP checksum, following a call to <code>tcp decode</code> . Note: this field is only valid after a decode operation. (<i>default = 00 00</i>) If <code>useValidChecksum</code> is set to <code>override</code> , the header checksum is a user-defined 2-byte hex value.
destPort	Protocol source port number. (<i>default = 0</i>)
finished <i>true/false</i>	The sender indicates that this is the last packet it transmits for the connection. (<i>default = false</i>)
offset	Offset from the beginning of the TCP header to the data. (<i>default = 5</i>)
pushFunctionValid <i>true/false</i>	Request that receiver deliver the packet to the application without buffering. (<i>default = false</i>)
resetConnection <i>true/false</i>	Reset the connection signal. (<i>default = false</i>)
sequenceNumber	Sequence number used to keep track of each byte of data. (<i>default = 0</i>)
sourcePort	Protocol destination port number. (<i>default = 0</i>)

synchronize <i>true/false</i>	Indicates either a connection request (ACK=0) or a connection accepted (ACK=1) condition. (<i>default = false</i>)
urgentPointer	Byte offset of the urgat data in the packet. (<i>default = 0</i>)
urgentPointerValid <i>true/false</i>	Indicates whether the urgent point field is valid. (<i>default = false</i>)
useValidChecksum <i>valid/invalid/override</i>	If <i>portFeatureTcpIPv4ChecksumOverride</i> = true, then: Valid: (<i>default</i>) The calculated header checksum is automatically calculated. Invalid: The calculated header checksum is automatically calculated (with error). Override: The header checksum can be set to a user-defined, 2-byte hex value.
window	The number of bytes that the recipient may send to the sender, starting at the acknowledge byte. (<i>default = 0</i>)

DEPRECATED OPTIONS

options	This option has no affect.
----------------	----------------------------

COMMANDS	The tcp command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

tcp cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tcp** command.

tcp config *option value*

Modify the TCP configuration options of the port. If no *option* is specified, returns a list describing all of the available TCP options (see STANDARD OPTIONS) for port.

tcp decode *capFrame [chasID cardID portID]*

Decodes a captured frame in the capture buffer and updates TclHal. **tcp cget** *option* command can be used after decoding to get the option data.

tcp get *chasID cardID portID*

Gets the current TCP configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Note that stream get must be called before this command's get sub-command. Call this command before calling **tcp cget** *option* to get the value of the configuration option.

tcp set *chasID cardID portID*

Sets the TCP configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **tcp config** *option value* command.

tcp setDefault

Sets to IxTclHal default values for all configuration options.

options

Variable length option field in the TCP header. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. (*default = {}*)

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1
set portList [list [list $chas $card $port]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

set portMAC {00 00 00 01 01 01}
set portIP {192.168.18.1}
set portMask {255.255.255.0}

set destMAC {00 00 00 01 01 02}
set destIP {192.168.18.2}
set destMask {255.255.255.0}

port setFactoryDefaults $chas $card $port
port setDefault

# Stream: 256 packets
stream setDefault
stream config -numFrames 256
```

```
stream config -sa $portMAC
stream config -da $destMAC
stream config -dma stopStream

# Set up IP: lowcost packets
# Source address varies by incrementing the network part
# Destination address varies by incrementing the host part
ip setDefault
ip config -cost lowCost
ip config -sourceIpAddr $portIP
ip config -sourceIpMask $portMask
ip config -sourceClass classC
ip config -destIpAddr $destIP
ip config -destIpMask $destMask
ip config -destClass classC
ip config -qosMode ipV4ConfigDscp
ip config -dscpMode ipV4DscpClassSelector
ip config -classSelector ipV4DscpClass2
ip set $chas $card $port

protocol setDefault
protocol config -name ipV4
protocol config -ethernetType ethernetII

tcp setDefault
tcp config -sourcePort 32768
tcp config -destPort 21
tcp set $chas $card $port

stream set $chas $card $port 1
port set $chas $card $port

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

stream, protocol, ip

NAME - tcpRoundTripFlow

tcpRoundTripFlow - configure the tcp round trip flow parameters for a port on a card on a chassis

SYNOPSIS

tcpRoundTripFlow *sub-command options*

DESCRIPTION

The **tcpRoundTripFlow** command is used to configure the tcp round trip flow specific information used when setting the tcp round trip flow on a port.

STANDARD OPTIONS

dataPattern *type*

Sets up the default data pattern to be inserted into the streams on the port. *type* may be one of the following values:

Option	Value	Usage
<i>allOnes</i>	0	the frame contains all 1's
<i>allZeroes</i>	1	the frame contains all 0's
<i>xAAAA</i>	2	the frame contains all A's
<i>x5555</i>	3	the frame contains all 5's
<i>x7777</i>	4	the frame contains all 7's
<i>xDDDD</i>	5	the frame contains all D's
<i>xFOFO</i>	6	the frame contains repeating pattern of F0F0's
<i>x0F0F</i>	7	the frame contains repeating pattern of 0F0F's
<i>xFF00FF00</i>	8	the frame contains repeating pattern of FF00FF00's
<i>x00FF00FF</i>	9	the frame contains repeating pattern of 00FF00FF's
<i>xFFFFF0000</i>	10	the frame contains repeating pattern of FFFF0000's
<i>x0000FFFF</i>	11	the frame contains repeating pattern of 0000FFFF's
<i>x00010203</i>	12	(<i>default</i>) the frame contains repeating pattern of 00010203's
<i>x00010002</i>	13	the frame contains repeating pattern of 00010002's
<i>xFFFEFD_C</i>	14	the frame contains repeating pattern of FFFEFDFC's
<i>xFFFFFFFE</i>	15	the frame contains repeating pattern of FFFFFFFE's
<i>userpattern</i>	16	select this type to insert user-defined data pattern in the frame, as defined in <i>pattern</i>

forceIpSA *true/false*

Forces the IP source address in reflected packets, as defined in the *ipSA* option. (*default = false*)

framesize

Number of bytes in each frame in the tcp round trip flow. (*default = 64*)

gatewayIpAddr

Gateway IP address. (*default = 0.0.0.0*)

ipSA

IP source address. (*default = 0.0.0.0*)

macDA

MAC destination address. (*default={00 00 00 00 00 00}*)

macSA

MAC source address. (*default={00 00 00 00 00 00}*)

pattern

Specify a user-defined pattern of data to be transmitted on this stream. The *dataPattern* option must be set to type *userpattern* or this pattern is ignored (*default= {00 01 02 03}*)

patternType *type*

Type of given patterns that is inserted in all the frames transmitted on the tcp round trip flow stream. *type* can be one of the following:

Option	Value	Usage
<i>incrByte</i>	0	increment each byte of the frame during transmission (<i>default</i>)
<i>incrWord</i>	1	increment each word of the frame during transmission
<i>decrByte</i>	2	decrement each byte of the frame during transmission
<i>decrWord</i>	3	decrement each word of the frame during
<i>patternTypeRandom</i>	4	generate random pattern of data during transmission
<i>repeat</i>	5	transmit the same pattern of data in the frame transmission
<i>nonRepeat</i>	6	transmit a fixed pattern of data. Note: <i>Fixed</i> type in IxExplorer.

useArpTable *true/false*

Enable ARP Mac destination address option. (*default = false*)

COMMANDS

The **tcpRoundTripFlow** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

tcpRoundTripFlow cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tcpRoundTripFlow** command.

tcpRoundTripFlow config *option value*

Modify the tcp round trip flow configuration options of the port. If no *option* is specified, returns a list describing all of the available the **tcpRoundTripFlow** options (see STANDARD OPTIONS) for port.

tcpRoundTripFlow get *chasID cardID portID*

Gets the current tcp round trip flow configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Call this command before calling **tcpRoundTripFlow** cget *option* to get the value of the configuration option.

tcpRoundTripFlow set *chasID cardID portID*

Sets the tcp round trip flow configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **tcpRoundTripFlow config *option value*** command.

tcpRoundTripFlow setDefault

Sets to IxTclHal default values for all configuration options.

tcpRoundTripFlow setFactoryDefaults chasID cardID portID

Sets the factory defaults to the tcpRoundTripFlow.

EXAMPLES

```

package require IxTclHal

# In this example, two ports on a 10/100 card are connected
# through a
# simple switch. The first port transmits at 100Mb/s and the
# second
# port transmits at 10Mb/s.
#
# The second port uses TCP Round Trip Flows to reflect the
# received
# packets back to port 1, where they are captured and analyzed for
# latency using captureBuffer.

# Connect to chassis and get chassis ID
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assumes that card 1 is a 10/100 card with both ports connected
# to
# a simple L2 switch
set card 1
set txPort 1
set rxPort 2

# Useful port lists
set portList [list [list $chas $card $txPort] [list $chas $card
$rxPort]]
set txPortList [list [list $chas $card $txPort]]
set rxPortList [list [list $chas $card $rxPort]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
# Test parameters
# MAC addresses
set p1MAC [list 00 00 00 01 01 01]

```

```

set p2MAC [list 00 00 00 01 01 02]
# IP addresses
set p1IP "192.168.18.1"
set p2IP "192.168.18.2"
# Number of frames to transmit
set numFrames 10

# Set up Transmit Port

# Port 1: 100Mb/s
port setFactoryDefaults $chassis $card $txPort
port config -speed 100
port config -advertise100FullDuplex true
port config -advertise100HalfDuplex false
port config -advertise10FullDuplex false
port config -advertise10HalfDuplex false

# Stream: 1 stream @ 100%, frame size 100, specific number of
frames
# Make sure to insert time stamps (fir)
stream setDefault
stream config -enable true
stream config -dma stopStream
stream config -numBursts 1
stream config -numFrames $numFrames
stream config -rateMode usePercentRate
stream config -percentPacketRate 100
stream config -sa $p1MAC
stream config -da $p2MAC
stream config -framesize 100
stream config -fir true

# IP: ethernetII tcp packets from port to port
ip setDefault
ip config -ipProtocol tcp
ip config -sourceIpAddr $p1IP
ip config -sourceIpAddrRepeatCount 1
ip config -sourceIpAddrMode fixed
ip config -destIpAddr $p2IP
ip config -destIpAddrRepeatCount 1
ip config -destIpAddrMode fixed
ip set $chassis $card $txPort

tcp setDefault
tcp set $chassis $card $txPort

protocol setDefault
protocol config -name ipv4
protocol config -ethernetType ethernetII

# Set the stream and ports
stream set $chassis $card $txPort 1
port set $chassis $card $txPort

# Set up Receive Port

# Port 2: 10Mb/s, TCP round trip mode reflects 64 byte packets
port setFactoryDefaults $chassis $card $rxPort
port setDefault
port config -speed 10
port config -advertise100FullDuplex false
port config -advertise100HalfDuplex false
port config -advertise10FullDuplex true

```

```

port config -advertise10HalfDuplex false
port config -transmitMode           portTxPacketFlows
port config -receiveMode          portRxTcpRoundTrip

# Set up TCP RT for Mac addresses
tcpRoundTripFlow setDefault
tcpRoundTripFlow config -macSA    $p2MAC
tcpRoundTripFlow config -macDA    $p1MAC
tcpRoundTripFlow set $chas $card $rxPort

# Set the port
port set $chas $card $rxPort
ixWritePortsToHardware portList

# Wait for changes to take affect
after 1000
ixCheckLinkState portList

# Send the packets and wait for things to be done
ixClearStats txPortList
ixStartCapture txPortList
ixStartTransmit txPortList

after 1000
ixCheckTransmitDone txPortList

# Fill the capture buffer with all of the packets
capture get $chas $card $txPort
set numRxFrames [capture cget -nPkts]
if {$numRxFrames != $numFrames} {
    ixPuts "$numFrames transmitted, but $numRxFrames received"
}

captureBuffer get $chas $card $txPort 1 [expr $numRxFrames - 1]

# Figure out the latency and print it out
captureBuffer getStatistics
captureBuffer getConstraint 1
ixPuts -nonewline "Avg Latency is "
ixPuts -nonewline [captureBuffer cget -averageLatency]
ixPuts -nonewline "ns, min = "
ixPuts -nonewline [captureBuffer cget -minLatency]
ixPuts -nonewline "ns, max = "
ixPuts -nonewline [captureBuffer cget -maxLatency]
ixPuts "ns"

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*stream, ip, tcp*

NAME - timeServer

timeServer - configure the timing parameters for a chassis.

SYNOPSIS

timeServer *sub-command options*

DESCRIPTION

The **timeServer** command is used to manage the timing of the chassis chain. It includes controls and read-only values for all timing options available on IXIA 100 chassis.

STANDARD OPTIONS

antennaStatus

Read-only. Possible values include:

Option	Value	Usage
<i>gpsStateAntennaUnknown</i>	0	antenna status is <i>unknown</i> until status is received from the GPS unit
<i>gpsStateAntennaOK</i>	1	antenna is connected and working
<i>gpsStateAntennaOpen</i>	2	antenna is not detected
<i>gpsStateAntennaShort</i>	3	antenna is not working

cdmaFrameErrorRate

Read-only. The CDMA frame error rate, expressed in errored frames per second.

cdmaSNR

Read-only. The CDMA signal to noise ratio.

cdmaState

Read-only. The current state of the CDMA unit. Possible values include:

Option	Value	Usage
<i>cdmaStateUnknown</i>	0	CDMA status is unknown until status is received.
<i>cdmaStateAcquiring</i>	1	acquiring a signal
<i>cdmaStateSignalDetected</i>	2	a CDMA signal has been detected
<i>cdmaStateCodeLocking</i>	3	CDMA code locking in progress
<i>cdmaStateCarrierLocking</i>	4	CDMA carrier locking in progress
<i>cdmaStateLocked</i>	5	CDMA code and carrier are locked; valid times are available

cdmaTime

Read-only. CDMA generated time in seconds.

**enableValidStats
*true / false***

If set, then *timeServer cget -statName* calls for statistics invalid for the time source returns an error. If unset, then all *timeServer cget -statName* returns without error, but the invalid statistics have default values. (*default = true*)

fpgaVersion

Read-only for GPS Receiver only. The version number of the GPS receiver FPGA.

gpsStatus

Read-only. Possible values include:

Option	Value	Usage
<i>gpsStateGpsUnknown</i>	0	GPS status is unknown until status is received
<i>gpsStateGpsLocked</i>	1	connection to the GPS is established

Option	Value	Usage
<code>gpsStateGpsUnlocked</code>	2	connection to the GPS is not established

gpsTime

Read-only. GPS generated time in seconds.

lockStatus

Read-only. For the GPS receiver only, shows the lock status of the chassis. One of::

Option	Value	Usage
<code>gpsUnlocked</code>	0	Chassis is not locked to the GPS receiver.
<code>gpsLocked</code>	1	Chassis is locked to the GPS receiver.

positionFix

Read-only for GPS Receiver only. The type of GPS signal received. Possible values include:

Option	Value	Usage
<code>gpsPositionInvalid</code>	0	No signal
<code>gpsPositionValidSPS</code>	1	SPS
<code>gpsPositionValidDGPS</code>	2	DGPS
<code>gpsPositionValidPPS</code>	3	PPS

pllStatus

Read-only. Possible values include:

Option	Value	Usage
<code>gpsStatePLLUnknown</code>	0	PLL status is unknown until status is received
<code>gpsStatePLLOK</code>	1	PLL is locked
<code>gpsStatePLLUnlocked</code>	2	PLL is not synchronized to the satellite

qualityStatus

Read-only. Possible values include:

Option	Value	Usage
<code>tsTimeQualityInvalid</code>	0	quality invalid until status is received
<code>tsTimeQuality0</code>	1	perfect timing
<code>tsTimeQuality1</code>	2	acceptable timing
<code>tsTimeQuality2</code>	3	not acceptable timing
<code>tsTimeQuality3</code>	4	not acceptable timing
<code>tsTimeQuality4</code>	5	not acceptable timing

satelliteIdRatios

Read-only for GPS Receiver only. The connection ratios of signal to noise for the first four satellites used.

satellitesUsed

Read-only for GPS Receiver only. The number of GPS satellites the receiver is connected to.

sntpClient

The name or IP address of the SNTP server used to obtain time information from. Used when `timeSource` is set to `sntpClient`. (`default = “”`)

state

Read-only. The current state of the GPS unit expressed as a string.

timeSource

Indicates the source for the time server:

Option	Value	Usage
tsInternal	0	(default) use internal timing for chassis.
tsGpsServer	1	use the GPS unit.
tsCdma	8	use the CDMA unit
tsGpsAfd1Server	9	use the GPS receiver

utcDate

Read-only for GPS Receiver only. The current date, in UTC form, expressed as a string.

utcTime

Read-only for GPS Receiver only. The current time of day, in UTC form, expressed as a string.

CThe following time source options are changed to *tsInternal* if used:

Option	Value	Usage
tsSntpServer	2	use an external SNTP server in sntpClient.
tsPcClock	3	use the clock from the PC associated with the chassis.
tsE1	4	(<i>IxClock only</i>) use the E1 clock input
tsT1	5	(<i>IxClock only</i>) use the T1 clock input
ts1PPS	6	(<i>IxClock only</i>) use the 1PPS clock input
tsStandAlone	7	use stand-alone timing for the chassis

DEPRECATED OPTIONS
e1T1Status

Read-only for IxClock only (obsolete). The status of the E1 or T1 signal. Possible values include:

Option	Value	Usage
<i>ixClockE1T1None</i>	0	no signal is detected
<i>ixClockE1T1Error</i>	1	an error has been detected
<i>ixClockE1T1OK</i>	2	signal is OK

timeOfDay

Read-only for IxClock only (obsolete). The current time of day, expressed as a string.

COMMANDS

The **timeServer** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

timeServer cget option

Returns the current value of the configuration option given by *option*.

timeServer config option value

Modify the configuration options of the time server. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for **timeServer**.

timeServer get chasID

Gets the current configuration of the TimeServer for chassis with chassis ID *chasID* from its hardware. Call this command before calling **timeServer cget option value** to get the value of the configuration option.

timeServer get *ipAddress*

Gets the current configuration of the TimeServer for the chassis whose IP address or hostname is *ipAddress*. Call this command before calling **timeServer cget option value** to get the value of the configuration option.

timeServer resetGps *chasID*

Resets the GPS unit in chassis ID *chasID*.

timeServer set *chasID*

Sets the time server configuration of the chassis with chassis ID *chasID* by reading the configuration option values set by the **timeServer config option value** command.

timeServer set *ipAddress*

Sets the time server configuration of the chassis whose IP address or hostname is *ipAddress* by reading the configuration option values set by the **timeServer config option value** command.

timeServer setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

set host cucumber
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Get the type of chassis so that we use GPS correctly
chassis get $host
set type [chassis cget -type]
# Set the time server selection to the default
timeServer setDefault
timeServer set $chas
chassis writeAll $chas

# And check the current settings
timeServer get $chas
set ts [timeServer cget -timeSource]
ixPuts -nonewline "Default time source is: "
switch $ts \
    $::tsInternal      {ixPuts "internal"} \
    $::tsGpsServer     {ixPuts "GPS"} \
```

```

$::tsSntpServer           {ixPuts "SNTP"} \
$::tsStandAlone           {ixPuts "stand alone"} \
$::tsCdma                 {ixPuts "CDMA"} \

# If the chassis is of a type that has GPS
if {$type == $::ixia100} \
{
    # Set it to GPS mode
    timeServer config -timeSource tsGpsServer
    timeServer set $chassis
    chassis writeAll $chassis

    # Wait for a minute to see if we can achieve good quality
    for {set i 0} {$i < 60} {incr i} \
    {
        after 1000

        # Get the settings
        timeServer get $chassis
        # Get the GPS time quality
        set quality [timeServer cget -qualityStatus]
        # If it's good enough
        if {$quality <= $::tsTimeQuality1} \
        {
            ixPuts "Good GPS quality achieved"
            break
        }
        # Otherwise report on all settings
        ixPuts "Quality is $quality"

        set quality [timeServer cget -antennaStatus]
        ixPuts "Antenna Status is $quality"

        set quality [timeServer cget -gpsStatus]
        ixPuts "GPS Status is $quality"

        set quality [timeServer cget -pllStatus]
        ixPuts "PLL Status is $quality"

        set quality [timeServer cget -state]
        ixPuts "State is $quality"
    }

    # If we achieved lock
    if {$i < 60} \
    {
        # Pick up the time setting
        set time [timeServer cget -gpsTime]
        ixPuts "Current time from GPS is $time"
    } \
    else \
    {
        ixPuts "Can't achieve GPS lock"
        break
    }
}

# Now try to set the system to use CDMA
timeServer config -timeSource tsCdma
timeServer set $chassis
chassis writeAll $chassis

```

SEE ALSO

chassisChain

NAME - txLane

txLane - configures and applies the lane skew configuration to the tx port.

SYNOPSIS	txLane <i>sub-command options</i>
DESCRIPTION	<p>The txLane command is used to configure and apply the lane skew configuration to the tx port.</p> <p>Users of this api should apply this config by the tcl command:</p> <ul style="list-style-type: none"> • <i>ixWriteConfigToHardware</i> <p>to not disurb the link state of the port on commit to hardware.</p>
<hr/>	
STANDARD OPTIONS	
pcsLane	Valid values range 0–19 for 100GB load modules; 0–3 for 40GB load modules. Negative testing allowed, so the physical lanes do not have to each have a unique value for pcsLane.. The method txLane mapping \$type overwrites any previously configured pcsLane setting. (<i>default = 1</i>)
skew	Value of the skew; this number is rounded up/down to the nearest actual skew the hardware supports. (<i>default = 0</i>)
<hr/>	
COMMANDS	<p>The <i>pcsLaneError</i> command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.</p> <p>txLane getLane <i>physicalLane</i> Retrieves one row in the tx lane indexed by the physical lane.</p> <p>txLane getLaneList <i>chassisID cardID portID</i> Utility method that returns a string, in tcl list form, of all the names of the physical lanes for this port in this configuration. Specific errors are:</p> <ul style="list-style-type: none"> • No connection to a chassis • Invalid port number <p>txLane setLane <i>physicalLane</i> Updates the row in the tx lane table with the configuration data for that physical lane.</p> <p>txLane select <i>chassisID cardID portID</i> Selects the local IxHAL object with the configured tx lane table. Does not apply directly to hardware; <i>ixWriteConfigToHardware</i> is required for commit to hardware. Required for any setLane/getLane operations. <i>setDefault</i> does not affect the selected port value. Specific errors are:</p> <ul style="list-style-type: none"> • No connection to a chassis • Invalid port number <p>txLane setDefault Sets to IxTclHal default values for all configuration options.</p>

EXAMPLES

See [*pcsLaneStatistics*](#)

SEE ALSO

[*pcsLaneStatistics*](#), [*pcsLaneError*](#)

NAME - txRxPreamble

txRxPreamble - configure the transmit and received preamble settings for 10GE LAN ports.

SYNOPSIS

txRxPreamble *sub-command options*

DESCRIPTION

The *txRxPreamble* command is used to set the options related to preamble transmit and receive operation on 10GE ports. Two of the options (*enableCiscoCDL* and *enableCDLStats*) apply to the use of the Cisco Converged Data Layer (CDL) on 10GE ports. The *enablePreambleView* option controls the ability to view the preamble in the result of a *stream cget -packetView* command.

STANDARD OPTIONS

enableCiscoCDL
true / false

This option enables the use of a Cisco CDL preamble to replace the standard Ethernet preamble. This feature is only available on some ports, which can be checked by a call to *port is ValidFeature... portFeatureCiscoCDL*. The contents of the preamble are programmed through the use of the *cdlPreamble* command. (*default = false*)

enableCDLStats
true / false

This option enables the generation of preamble statistics and capture. This feature is only available on some ports, which can be checked by a call to *port is ValidFeature... portFeaturePreambleCapture*. The statistics are available through the use of the *stat* command and the captured data is available through the use of the *capture* and *captureBuffer* commands.

enableIncludePreamble
InRxCrc
true / false

This option enables the inclusion of the preamble length in the receive side CRC calculation. (*default = false*)

enablePreambleView
true / false

This option enables the inclusion of the preamble in the *packetView* option of the *stream* command. This feature is only available on some ports, which can be checked by a call to *port is ValidFeature... portFeaturePreambleView*. (*default = 0*)

rxMode

The receive mode for the port.

Option	Value	Usage
<i>preambleModeSFDetect</i>	0	The SFD is the last byte in the preamble (the 8th byte in this case). This mode checks for the first occurrence of the SFD byte. The next byte is considered the start of the frame.
<i>preambleByteCount</i>	1	This mode counts the bytes of the preamble (8 bytes in this case), and considers the next byte (9th) the first byte of the frame.
<i>preambleSameAsTransmit</i>	2	(<i>default</i>) The Receive side accepts the same choices/entries that were made for the Transmit side.

txMode

The transmit mode for the port.

Option	Value	Usage
<i>preambleModeSFDetect</i>	0	(default) The SFD is the last byte in the preamble (the 8th byte in this case). This mode checks for the first occurrence of the SFD byte. The next byte is considered the start of the frame.
<i>preambleByteCount</i>	1	This mode counts the bytes of the preamble (8 bytes in this case), and considers the next byte (9th) the first byte of the frame.

COMMANDS

The *txRxPreamble* command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

txRxPreamble cget *option*

Returns the current value of the configuration option *txRxPreamble* by *option*. *Option* may have any of the values accepted by the **txRxPreamble** command, subject to the setting of the *enableValidStats* option.

txRxPreamble config *option value*

Modify the configuration options of the time server. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for txRxPreamble.

txRxPreamble get *chasID cardID portID*

Gets the current preamble configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Call this command before calling *txRxPreamble cget option* to get the value of the configuration option.

txRxPreamble set *chasID cardID portID*

Sets the preamble configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the *txRxPreamble config option value* command.

txRxPreamble setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
set chasID 1
set cardID 4
set portID 1

txRxPreamble setDefault
txRxPreamble config -rxMode preambleByteCount
txRxPreamble config -txMode preambleModeSFDetect
if [port isValidFeature $chasID $cardID $portID / 
portFeatureCiscoCDL] {
    txRxPreamble config -enableCiscoCDL true
}
if [port isValidFeature chasID cardID portID
portFeaturePreambleCapture] {
    txRxPreamble config -enableCDLStats true
}
if [port isValidFeature chasID cardID portID
portFeaturePreambleView] {
    txRxPreamble config -enablePreambleView true
}
```

A

txRxPreamble

}

SEE ALSO

stream, *cdlPreamble*

NAME - udf

udf - configure the User-Definable Fields in the frames of a stream.

SYNOPSIS

udf sub-command options

DESCRIPTION

User-Definable Fields (UDFs) are counters that can be inserted anywhere in the frame whose data can be used to represent special purpose patterns. Each of the supported UDFs can be enabled or disabled and contain 8, 16, 24, or 32 bit counters.

The *udf* command is used to configure the UDF parameters on a stream of a port. It must be followed by a call to *stream set*.

Table UDFs are a type of UDF which allows multiple static values to be placed at multiple locations in a packet. Table UDFs are enabled and controlled by the *tableUdf* and *tableUdfColumn* commands.

Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in *atmHeader*. The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command is with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

STANDARD OPTIONS**bitOffset**

Sets the offset of the UDF (in bits). This must be a value from 0-7 and is only supported on certain cards in certain modes. If this is set to a nonzero value when it is not legal, a 'stream set' error is issued.

cascadeType

Indicates the source of the initial value for the counter. The initial value for the first enabled stream always comes from the *initval* option.

Option	Value	Usage
<i>udfCascadeNone</i>	0	(default) The initial value always comes from <i>initval</i> .
<i>udfCascadeFromPrevious</i>	1	The initial value is derived from the last executed stream which used this UDF number with <i>cascadeType</i> set to <i>udfCascadeFromPrevious</i> . An initial increment/decrement/random operation is applied from the previous value.
<i>udfCascadeFromSelf</i>	2	The initial value is derived from the last value generated by this UDF with this stream. An initial increment/decrement/random operation is applied from the previous value.

chainFrom

Select what UDF the current UDF should chain from. When this option is employed, the UDF stays in its initial value until the UDF it is chained from reaches its terminating value. Values: None, UDF1 through UDF5 depending on

the number of UDFs available for the module, and excluding the UDF that is being configured.:

Option	Value	Usage
udfNone	0	(default)
udf1	1	chains from UDF1
udf2	2	chains from UDF2
udf3	3	chains from UDF3
udf4	4	chains from UDF4
udf5	5	chains from UDF5

continuousCount
true/false

When set to true, the counter increments or decrements the bytes depending on the *updown* option. (default = false)

counterMode

The mode of operation of the counter. The following values can be specified for this option:

Option	Value	Usage
udfCounterMode	0	(default) Normal up-down counter as controlled by <i>continuousCount</i> , <i>udfSize</i> , <i>initval</i> , <i>maskselect</i> , <i>maskval</i> , <i>random</i> , <i>repeat</i> , <i>step</i> , <i>updown</i> and <i>cascadeType</i> .
udfRandomMode	1	Generates random values, based on the values in <i>udfSize</i> , <i>maskselect</i> and <i>maskval</i>
udfValueListMode	2	A list of distinct values, based on the values of <i>udfSize</i> , <i>valueList</i> and <i>cascadeType</i> .
udfNestedCounterMode	3	Two nested counters may be used to build complex sequences, based on the values of <i>udfSize</i> , <i>initval</i> , <i>innerLoop</i> , <i>innerRepeat</i> , <i>innerStep</i> , <i>step</i> , <i>repeat</i> and <i>cascadeType</i> .
udfRangeListMode	4	A list of value ranges, based on <i>udfSize</i> , <i>cascadeType</i> and ranges. Ranges must be added to the <i>udf</i> command using the <i>addRange</i> sub-command.
udfIPv4Mode	5	A counter which facilitates generation of IPv4 addresses, based on <i>initval</i> , <i>innerRepeat</i> , <i>innerStep</i> , <i>continuousCount</i> , <i>repeat</i> , <i>enableSkipZerosAndOnes</i> and <i>skipMaskBits</i> .

Not all modes are supported by all port types and not all modes are supported by all UDFs on a port. A *stream set* fails if any enabled UDF does not support a *counterMode*. The availability of a particular mode on a particular UDF can be checked with the *port isValidFeature* command.

countertype

Earlier values of *countertype* are still valid but on boards and modes that support it, *countertype* is deprecated in favor of *udfSize*.

Describes the size and shape of this UDF field. Each field consists of 4 8-bit counters; these counters may be configured as individual counters or in any combination, such as 2 8-bit counters & one 16 bit counter, 2 16-bit counters, or 1 32 bit counter. Note that every 8-bit counter within this field does not have to be used. The options available for this variable select the size (8, 16, 24 or 32 bits) and configuration; for example - if the option c8x8x8x8 is selected the counters is configured as 4 independent 8-bit counters. If the option config8x16 is

selected, the counters is configured as one 8-bit counter, one 16-bit counter and the remaining 8-bits is unused. The following values can be specified for this option:

Option	Value	Usage
<i>c8</i>	0	(default) one 8-bit counter
<i>c16</i>	1	one 16 bit counter
<i>c8x8</i>	2	two 8-bit counters
<i>c24</i>	3	one 24-bit counter
<i>c16x8</i>	4	one 16-bit counter followed by a 8-bit counter
<i>c8x16</i>	5	one 8-bit counter followed by a 16-bit counter
<i>c8x8x8</i>	6	three 8-bit counters
<i>c32</i>	7	one 32-bit counter
<i>c24x8</i>	8	one 24-bit counter followed by a 8-bit counter
<i>c16x16</i>	9	two 16-bit counters
<i>c16x8x8</i>	10	one 16-bit counter followed by two 8-bit counters
<i>c8x24</i>	11	one 8-bit counter followed by a 24-bit counter
<i>c8x16x8</i>	12	one 8-bit counter followed by a 16-bit counter followed by another 8-bit counter
<i>c8x8x16</i>	13	two 8-bit counters followed by a 16-bit counter
<i>c8x8x8x8</i>	14	four 8-bit counters

enable *true/false*

If this option is set to *true*, then this UDF counter is inserted into the frame.
(*default = false*)

**enableCascade
*true/false***

If this option is set to *true*, then the UDF counter is **not** reset with the start of each stream, but rather continues counting from the ending value of the previous stream. (*default = false*)

enableIndexMode

If this option is set to *true*, the index mode is enabled.

enableKillBitMode

If this option is set to true, enables Kill Bit Mode.

killBitUDFSize

The Kill Bit UDF size.

**enableSkipZeros
AndOnes**

If *counterMode* is *udfIPv4Mode* and this option is set to *true*, then values of all 0's and all 1's as masked by *skipMaskBits* is skipped when generating values. This normally corresponds to network broadcast addresses. (*default = false*)

initval

The initial value of the counter. (*default = {08 00}*)

The default value in Tcl is different than the default value in IxExplorer GUI.

innerLoop

The number of times the inner loop is repeated. Used when *counterMode* is set to *udfNestedCounterMode*. (*default = 1*)

valueRepeatCount

The repeat count for each valuelist udf entry.

innerRepeat

The number of times each value in the inner loop is repeated. Used when *counterMode* is set to *udfNestedCounterMode*. (*default = 1*)

innerStep	The steps size between inner loop values. Used when <i>counterMode</i> is set to <i>udfNestedCounterMode</i> . (<i>default = 1</i>)
linearCoefficientEnable	Enables the linear coefficient.
linearCoefficient	The linear coefficient value.
linearCoefficientLoopCount0	The value of coefficient loop count is 0.
linearCoefficientLoopCount2	The value of coefficient loop count is 2.
tripleNestedLoop0Increment	The triple nested loop increment value is set to 0.
maskselect	This is a 32-bit mask that enables, on a bit-by-bit basis, use of the absolute counter value bits as defined by <i>maskval</i> option. (<i>default = {00 00}</i>)
maskval	A 32-bit mask of absolute values for this UDF counter. It is used in association with the <i>maskselect</i> ; bits must be set 'on' or the bits in <i>maskselect</i> is ignored. (<i>default = {00 00}</i>)
offset	The absolute offset to insert this udf into the frame. Note that DA and SA use the fixed offsets at 0 and 6, respectively. This option applies to all <i>counterModes</i> . (<i>default = 12</i>)
random true/false	If this object is set to <i>true</i> , then this counter contains random data. The UDFs may not have part counter and part random data. (<i>default = false</i>)
repeat	The counter is incremented or decremented the number of times based on this option. If <i>continuousCount</i> option is set then this value is ignored. (<i>default = 1</i>)
skipMaskBits	If <i>counterMode</i> is <i>udfIPv4Mode</i> and <i>enableSkipZerosAndOnes</i> is set to <i>true</i> , this is the number of low order bits to check when looking for all 0's and all 1's. This normally corresponds to network broadcast addresses. (<i>default = 8</i>)
step	The step size for counter increment/decrement, if supported by the load module. (<i>default = 1</i>)
udfSize	Sets the UDF field size (in bits). This must be a value from 1-32 and is only supported on certain cards in certain modes. If this is set to a nonzero value when it is not legal, a 'stream set' error is issued.
updown	This option describes whether each of the 8-bit counters are to be incremented or decremented. If two or more counters are cascaded together as a larger counter (ie, 16,24 or 32-bit counter), that group of counters must all be incremented or decremented. Note that the most-significant byte selection takes precedence if there is a discrepancy. The possible values of this options are:

Option	Value	Usage
<i>uuuu</i>	15	(<i>default</i>) all bytes are incrementing
<i>uuud</i>	14	bytes 1,2 and 3 are incrementing and byte 4 is decrementing
<i>uudu</i>	13	bytes 1,2 and 4 are incrementing and byte 3 is decrementing

Option	Value	Usage
<i>uudd</i>	12	bytes 1 and 2 are incrementing and bytes 3 and 4 are decrementing
<i>uduu</i>	11	bytes 1,3 and 4 are incrementing and byte 2 is decrementing
<i>udud</i>	10	bytes 1 and 3 are incrementing and bytes 2 and 4 are decrementing
<i>uddu</i>	9	bytes 1 and 4 are incrementing and bytes 2 and 3 are decrementing
<i>uddd</i>	8	byte 1 is incrementing and bytes 2,3 and 4 are decrementing
<i>duuu</i>	7	byte 1 is decrementing and bytes 2,3 and 4 are incrementing
<i>duud</i>	6	bytes 1 and 4 are decrementing and bytes 2 and 3 are incrementing
<i>dudu</i>	5	bytes 1 and 3 are decrementing and bytes 2 and 4 are incrementing
<i>dudd</i>	4	bytes 1,3 and 4 are decrementing and byte 2 is incrementing
<i>dduu</i>	3	bytes 1 and 2 are decrementing and bytes 3 and 4 are incrementing
<i>ddud</i>	2	bytes 1,2 and 4 are decrementing and byte 3 is incrementing
<i>dddu</i>	1	bytes 1,2 and 3 are decrementing and byte 4 is incrementing
<i>dddd</i>	0	all bytes are decrementing

valueList

A list which holds the values to be used when *counterMode* is set to *udfValueListMode*. (*default* = {})

DEPRECATED OPTIONS
countertype

Earlier values of *countertype* are still valid but on boards and modes that support it, *countertype* is deprecated in favor of *udfSize*:

Option	Value	Usage
<i>c8</i>	0	(<i>default</i>) one 8-bit counter
<i>c16</i>	1	one 16 bit counter
<i>c8x8</i>	2	two 8-bit counters
<i>c24</i>	3	one 24-bit counter
<i>c16x8</i>	4	one 16-bit counter followed by a 8-bit counter
<i>c8x16</i>	5	one 8-bit counter followed by a 16-bit counter
<i>c8x8x8</i>	6	three 8-bit counters
<i>c32</i>	7	one 32-bit counter
<i>c24x8</i>	8	one 24-bit counter followed by a 8-bit counter
<i>c16x16</i>	9	two 16-bit counters
<i>c16x8x8</i>	10	one 16-bit counter followed by two 8-bit counters
<i>c8x24</i>	11	one 8-bit counter followed by a 24-bit counter
<i>c8x16x8</i>	12	one 8-bit counter followed by a 16-bit counter followed by another 8-bit counter
<i>c8x8x16</i>	13	two 8-bit counters followed by a 16-bit counter
<i>c8x8x8x8</i>	14	four 8-bit counters

COMMANDS

The **udf** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

udf addRange

Used when *counterMode* is set to *udfRangeListMode*. Adds the values in *initVal*, *repeat* and *step* to the list of values associated with the UDF. Ranges added to the range list are given an index starting at 1; this is used in the *getRange* sub-command. Specific errors are:

- Invalid UDF parameters

udf cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **udf** command.

udf clearRangeList

Clears all values in the range list associated with the UDF.

udf config option value

Modify the configuration options of the port. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for port.

udf get udfID

After using *stream get* command, this command gets the UDF with id *udfID*.

udf getFirstRange

Finds the first range in the range list and places the values in *initval*, *repeat* and *step*. Specific errors are:

- The list is empty.

udf getNextRange

Finds the next range in the range list and places the values in *initval*, *repeat* and *step*. *getFirstRange* must have been called before this call. Specific errors are:

- *getFirstRange* has not been called.

udf getRange rangeIndex

Finds the range in the range list with index *rangeIndex* and places the values in *initval*, *repeat* and *step*. Specific errors are:

- There is no object with this ID.

udf set udfID

Sets the configuration of the UDF with ID *udfID* by reading the configuration option values set by the **udf config option value** command. *stream set* must be called after setting this UDF.

Note: The command **stream setDefault** also overwrites the **udf set** command.

udf setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assume card 4 has a TXS4, with every UDF function
set card 68
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Make sure the port is at factory default
port setFactoryDefaults $chas $card $port
stream setDefault

# UDF 3: normal counter mode, 8 bits counting up continuously
# from 0
udf setDefault
udf config -enable true
udf config -offset 12
udf config -udfSize c8
udf config -counterMode udfCounterMode
udf config -continuousCount true
udf config -updown uuuu
udf config -initval 00
# Set UDF 3
udf set 3

# UDF 1: 24-bits at offset 12 in packet
# Two ranges: start = 0x4200, increment by 14, repeat 100
```

```

# start = 0x100000, increment by 100, repeat 2

# Remove all existing range list items
udf clearRangeList
udf setDefault
udf config -enable true
udf config -counterMode udfRangeListMode
udf config -offset 12
udf config -udfSize c24
udf config -initval {00 00 42 00}
udf config -repeat 100
udf config -step 14

# Add the range to the UDF
udf addRange

udf config -initval {00 10 00 00}
udf config -repeat 2
udf config -step 100

# Add the second range to the UDF
udf addRange

# Set UDF 1
udf set 1

# UDF 2: 8-bits at offset 12 in packet
# Value list mode. Values are: 0x01, 0x10, 0x42
udf setDefault
udf config -enable true
udf config -counterMode udfValueListMode
udf config -offset 12
udf config -udfSize c8
udf config -valueList { { 00 00 00 01 } \
                        { 00 00 00 10 } \
                        { 00 00 00 42 } }

# Set UDF 2
udf set 2

# UDF4: 16 bits at offset 12
# Nested counters: Outer: start at 0x0100, step by 10,
# repeat 100 times
# Inner: repeat each value 2 times,
# step by 4, repeat 3 times
udf setDefault
udf config -enable true
udf config -offset 12
udf config -udfSize c16
udf config -counterMode udfNestedCounterMode
udf config -initval {01 00}
udf config -repeat 100
udf config -step 10
udf config -innerRepeat 2
udf config -innerStep 4
udf config -innerLoop 3

# Set UDF 4
udf set 4

# Make sure to use stream set to set the UDFs
stream set $chas $card $port 1
ixWriteConfigToHardware portList

```

```
# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

stream, tableUdf, tableUdfColumn

NAME - udp

udp - configure the UDP parameters for a port on a card on a chassis

SYNOPSIS

udp *sub-command options*

DESCRIPTION

The **udp** command is used to configure the UDP-specific information used when building UDP type packets if **ip config -ipProtocol** has been set to *Udp*. See RFC 768 for a complete definition of UDP header fields. Note that **stream** get must be called before this command's get sub-command.

Note that when using ATM ports, different types of ATM encapsulation result in different length headers, as discussed in [atmHeader](#). The data portion of the packet normally follows the header, except in the case of the two LLC Bridged Ethernet choices, where 12 octets of MAC address and 2 octets of Ethernet type follow the header. The offsets used in this command are with respect to the beginning of the AAL5 packet and must be adjusted by hand to account for the header.

STANDARD OPTIONS

checksum

Value of the checksum in the valid udp stream. Valid only if the **stream set** is performed and **enableChecksumOverride** is true. (*default = {00 00}*)

checksumMode

Indicates whether a valid checksum should be inserted in the packet or not.

Option	Value	Usage
validChecksum	0	(<i>default</i>) a valid checksum is used
invalidChecksum	1	the checksum indicated in the checksum option is used

destPort

The port of the destination process. Well-known port values include:

Option	Value	Usage
echoServerPort	7	(<i>default</i>)
discardPacketPort	9	
usersServerPort	11	
dayAndTimeServerPort	13	
quoteOfTheDayServerPort	17	
characterGeneratorPort	19	
timeServerPort	37	
wholsServerPort	43	
domainNameServerPort	53	
unassignedPort	63	
bootpServerPort	67	
bootpClientPort	68	
tftpProtocolPort	69	

Option	Value	Usage
remoteWhoServerPort	513	
ripPort	520	
ptpEventPort	319	
ptpGeneralPort	320	

enableChecksum
true/false

If set to true, a valid UDP checksum is calculated for each frame. If set to false, the UDP checksum is invalid. (*default = false*)

enableChecksum
Override true/false

If set to *true*, the calculated checksum is replaced with the value in *checksum*. (*default = false*)

length

Length of the datagram including header and the data. (*default = 0*)

lengthOverride true/
false

Allows to change the length in udp header. (*default = false*)

sourcePort

The port of the sending process. Well-known port values include:

Option	Value	Usage
echoServerPort	7	(<i>default</i>)
discardPacketPort	9	
usersServerPort	11	
dayAndTimeServerPort	13	
quoteOfTheDayServerPort	17	
characterGeneratorPort	19	
timeServerPort	37	
wholsServerPort	43	
domainNameServerPort	53	
unassignedPort	63	
bootpServerPort	67	
bootpClientPort	68	
tftpProtocolPort	69	
remoteWhoServerPort	513	
ripPort	520	
ptpEventPort	319	
ptpGeneralPort	320	

DEPRECATED
OPTIONS**enableChecksum**

This option is deprecated.

COMMANDS

The **udp** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

udp cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **udp** command.

udp config option value

Modify the UDP configuration options of the port. If no *option* is specified, returns a list describing all of the available UDP options (see STANDARD OPTIONS) for port.

udp decode capFrame [chassisID cardID portID]

Decodes a captured frame in the capture buffer and updates TclHal. **udp cget option** command can be used after decoding to get the option data.

udp get chassisID cardID portID

Gets the current UDP configuration of the port with id *portID* on card *cardID*, chassis *chassisID*. Note that *stream* get must be called before this command's get sub-command. Call this command before calling **udp cget option** to get the value of the configuration option.

udp set chassisID cardID portID

Sets the UDP configuration of the indicated port by reading the configuration option values set by the **udp config option value** command.

udp setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assume card 4 has a TXS4, with every UDF function
set card 68
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
```

```
        return 1
    }
    # Take ownership of the ports we'll use
    if [ixTakeOwnership $portList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Make sure the port is at factory default
    port setFactoryDefaults $chassis $card $port
    stream setDefault

    # UDF 3: normal counter mode, 8 bits counting up continuously
    # from 0
    udf   setDefault
    udf    config   -enable      true
    udf    config   -offset      12
    udf    config   -udfSize    c8
    udf    config   -counterMode udfCounterMode
    udf    config   -continuousCount true
    udf    config   -updown     uuuu
    udf    config   -initval    00
    # Set UDF 3
    udf    set      3

    # UDF 1: 24-bits at offset 12 in packet
    # Two ranges: start = 0x4200, increment by 14, repeat 100
    #           start = 0x100000, increment by 100, repeat 2

    # Remove all existing range list items
    udf    clearRangeList
    udf   setDefault
    udf    config   -enable      true
    udf    config   -counterMode udfRangeListMode
    udf    config   -offset      12
    udf    config   -udfSize    c24
    udf    config   -initval    {00 00 42 00}
    udf    config   -repeat     100
    udf    config   -step       14

    # Add the range to the UDF
    udf    addRange

    udf    config   -initval    {00 10 00 00}
    udf    config   -repeat     2
    udf    config   -step       100

    # Add the second range to the UDF
    udf    addRange

    # Set UDF 1
    udf    set      1

    # UDF 2: 8-bits at offset 12 in packet
    # Value list mode. Values are: 0x01, 0x10, 0x42
    udf   setDefault
    udf    config   -enable      true
    udf    config   -counterMode udfValueListMode
    udf    config   -offset      12
    udf    config   -udfSize    c8
    udf    config   -valueList   { { 00 00 00 01 } \
                                { 00 00 00 10 } \
                                { 00 00 00 42 } }
```

```
# Set UDF 2
udf    set        2

# UDF4: 16 bits at offset 12
# Nested counters: Outer: start at 0x0100, step by 10,
#                      repeat 100 times
#                      Inner: repeat each value 2 times,
#                             step by 4, repeat 3 times
udf    setDefault
udf    config   -enable      true
udf    config   -offset      12
udf    config   -udfSize    c16
udf    config   -counterMode udfNestedCounterMode
udf    config   -initval    {01 00}
udf    config   -repeat     100
udf    config   -step       10
udf    config   -innerRepeat 2
udf    config   -innerStep   4
udf    config   -innerLoop   3

# Set UDF 4
udf    set        4

# Make sure to use stream set to set the UDFs
stream set $chassis $card $port 1
ixWriteConfigToHardware      portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

stream, protocol, ip

NAME - usb

usb - view the properties of a USB port of a card on a chassis.

Note: THIS COMMAND IS DEPRECATED IN ITS ENTIRETY.

SYNOPSIS	<code>usb sub-command options</code>
DESCRIPTION	The usb command is used to view the properties of a USB port of a card on a chassis.
<hr/>	
STANDARD OPTIONS	
cpeMacAddress	<i>Read-only</i> . The MAC address of the CPE (Customer Premise Equipment).
deviceClass	<i>Read-only</i> . Class of the attached device, according to the document: <i>Universal Serial Bus Class Definitions for Communication Devices Version 1.1 January 19, 1999</i> .
ethernetMaxSegmentSize	<i>Read-only</i> . The maximum Ethernet segment size.
manufacturer	<i>Read-only</i> . Manufacturer of the attached device.
maxUSBPacketSize	<i>Read-only</i> . The maximum size of the USB packets. Either: In 64 bytes, out 64 bytes (<i>0</i>) In 32 bytes, out 32 bytes (<i>1</i>)
product	<i>Read-only</i> . The product name of the device which is attached.
productID	<i>Read-only</i> . The product identification number of the attached device.
releaseNumber	<i>Read-only</i> . Release level of USB supported by the attached device.
serialNumber	<i>Read-only</i> . Serial number of the attached device.
vendorID	<i>Read-only</i> . Select this radio button to put this module into USB mode.
<hr/>	
COMMANDS	The usb command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

usb get chasID cardID portID

Gets the current configuration of the port with id *portID* on card *cardID*, chassis *chasID*. from its hardware. Call this command before calling **usb cget option value** to get the value of the configuration option. Specific errors are:

- No connection to a chassis
- Invalid port number
- The port is not a Usb port.

usb reset *chasID cardID portID*

Sends a reset signal to the device on the port with id *portID* on card *cardID*, chassis *chasID*.

usb setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

SEE ALSO

card, port

NAME - **version**

version - get version information for IxTclHal.

SYNOPSIS

version *sub-command options*

DESCRIPTION

This command allows to view the version information for IxTclHal. Note that when using TCL from a Unix system, the version may not be obtained until a connection to the chassis is made, for example through the use of *ixConnectToChassis*.

STANDARD OPTIONS

companyName	<i>Read-only</i> . The name of company: Ixia Communications
copyright	<i>Read-only</i> . Copyright banner for IxTclHal
installVersion	<i>Read-only</i> . Installed version of the software.
ixTclHALVersion	<i>Read-only</i> . The version number of ixTclHal.dll file
ixTclProtocolVersion	<i>Read-only</i> . The version of the IxRouter protocol.
productVersion	<i>Read-only</i> . The software version along with build number

COMMANDS

The **version** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

version **cget** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **version** command.

version **config**

Modify the **version** configuration options of the IxTclHal. If no *option* is specified, returns a list describing all of the available version options (see STANDARD OPTIONS).

version **get**

Gets the current version information from HAL. Call this command before calling **version cget** *option* to get the value of the configuration option.

EXAMPLES

```
package require IxTclHal

version get

ixPuts -nonewline "Company name is "
ixPuts [version cget -companyName]
ixPuts -nonewline "Copyright is "
ixPuts [version cget -copyright]
```

```
ixPuts -nonewline "Install Version is "
ixPuts           [version cget -installVersion]
ixPuts -nonewline "ixTclHAL Version is "
ixPuts           [version cget -ixTclHALVersion]
ixPuts -nonewline "ixTclProtocolVersion is "
ixPuts           [version cget -ixTclProtocolVersion]
ixPuts -nonewline "Product Version is "
ixPuts           [version cget -productVersion]
```

SEE ALSO

NAME - VFTHeader

VFTHeader-sets up VFT Header over Fibre Channel.

SYNOPSIS

VFTHeader *sub-command options*

DESCRIPTION

The Virtual Fabric Tagging Header (VFT Header) allows Fibre Channel frames to be tagged with the Virtual Fabric Identifier (VF_ID) of the Virtual Fabric to which they belong. Tagged frames, that is frames with a VFT_Header, belonging to different Virtual Fabrics may be transmitted over the same physical link.

STANDARD OPTIONS

type

Specifies the kind of tagged frame. To use with Fibre Channel, type is set to 0. The use of other values is beyond the scope of this standard. No device sends a VFT tagged frame with a Type value in the VFT_Header other than 0h. A device receiving a VFT tagged frame with a Type value in the VFT_Header having a non-zero value discards the frame.

version

Specifies the version of the VFT Header. The default is 0.

routingControl

The R_CTL field is a one-byte field that contains routing bits and information bits to categorize the frame function.

The R_CTL is set to the value 50h to identify the VFT Extended Header.

hopCt

The count by which the VFT header packet is forwarded in the stream.

If the Hop Count Valid (HCV) bit is set to one, the Hop Count (Hop_Cnt) field specifies the number of hops remaining before the frame is discarded.

priority

Specifies the Quality of Service (QoS) value for the frame.

When set to zero, is interpreted to contain management information for the class of service.

virtualFabricId

The ID of the VFT header. It specifies the Virtual Fabric Identifier of the Virtual Fabric to which the tagged frame belongs.

COMMANDS

The **VFTHeader** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

VFTHeader setDefault

Returns the default settings.

EXAMPLES

See under *fibreChannel*.

SEE ALSO

fibreChannel

NAME - vlan

vlan - configure the VLAN parameters for a port on a card on a chassis

SYNOPSIS

vlan *sub-command options*

DESCRIPTION

The **vlan** command is used to configure the VLAN-specific information used when building 802.1q-type packets. See IEEE 802.1p/q for a complete definition of VLAN tag fields. It is enabled using *protocol config -enable802dot1qTag true*

STANDARD OPTIONS

cfi

Canonical Format Indicator is a single bit flag value. Options include:

Option	Value	Usage
<i>resetCFI</i>	0	(default) sets the CFI bit to low
<i>setCFI</i>	1	sets the CFI bit to high

maskval

When *mode* is set to *vCtrRandom*, this option indicates which bits of the VID counter may vary and which must remain constant. (default = 0000XXXXXXXXXXXX)

mode

Specifies how the vlanID tag is incremented or decremented. Only the top two VLAN elements in a stacked VLAN may used these values. Possible values include:

Option	Value	Usage
<i>vIdle</i>	0	(default) No change to VlanID tag regardless of repeat
<i>vIncrement</i>	1	The VlanID tag is incremented by <i>step</i> for <i>repeat</i> number of times.
<i>vDecrement</i>	2	The VlanID tag is decremented by <i>step</i> for <i>repeat</i> number of times.
<i>vContIncrement</i>	3	The VlanID tag is continuously incremented by <i>step</i> .
<i>vContDecrement</i>	4	The VlanID tag is continuously decremented by <i>step</i> .
<i>vCtrRandom</i>	5	Generate random VlanID tag for each frame
<i>vNestedIncrement</i>	100	For the second VLAN in a <i>stackedVlan</i> , this may be used to performed nested increment with respect to the first stack element.
<i>vNestedDecrement</i>	101	For the second VLAN in a <i>stackedVlan</i> , this may be used to performed nested decrement with respect to the first stack element.

name

Read-only. The name of the VLAN element, which may have been set in IxExplorer.

protocolTagId

The protocol ID field of the VLAN tag.

Option	Value	Usage
<i>vlanProtocolTag8100</i>	0x8100	(default)

Option	Value	Usage
vlanProtocolTag9100	0x9100	
vlanProtocolTag9200	0x9200	
vlanProtocolTag88A8	0x88A8	
vlanProtocolTag9300	0x9300	

repeat

The number of times the counter is to be repeated with the same value. If the *mode* option is set to *idle* then this value is ignored. Note that the repeat value is a 32-bit signed integer. (*default* = 10)

step

If *mode* is set to one of the increment or decrement settings, this is the step size between generated values. (*default* = 1)

userPriority

The user priority field is three bits in length, representing eight priority levels, 0 through 7. The use and interpretation of this field is defined in ISO/IEC 15802-3. (*default* = 0)

vlanID

The 12-bit VLAN Identifier (VID). (*default* = 0)

COMMANDS

The **vlan** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

vlan cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **vlan** command.

vlan config option value

Modify the vlan configuration options of the port. If no *option* is specified, returns a list describing all of the available vlan options (see STANDARD OPTIONS) for port.

vlan decode capFrame [chasID cardID portID]

Decodes a captured frame in the capture buffer and updates TclHal. **vlan cget option** command can be used after decoding to get the option data.

vlan get chasID cardID portID

Gets the current UDP configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Call this command before calling **vlan cget option** to get the value of the configuration option.

vlan set chasID cardID portID

Sets the vlan configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **vlan config option value** command.

vlan setDefault

Sets to IxTclHal default values for all configuration options.

EXAMPLES

```

package require IxTclHal

set host loopback

# Now connect to the chassis
if { [ixConnectToChassis $host] } {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [chassis cget -id]
set card 2
set port 4
set portList [list [list $chas $card $port]]

# Case 1: single VLAN
stream setDefault
protocol setDefault
protocol config -name          ipv4
protocol config -ethernetType   ethernetII
protocol config -enable802dot1qTag vlanSingle

vlan setDefault
vlan config -vlanID           42
vlan config -mode              vIncrement
vlan config -step              4
vlan config -repeat            10
if { [vlan set $chas $card $port] } {
    ixPuts $::ixErrorInfo
    return 1
}

if { [stream set $chas $card $port 1] } {
    ixPuts $::ixErrorInfo
    return 1
}

# Case 2: stacked VLAN with three elements
stream setDefault

protocol setDefault
protocol config -name          ipv4
protocol config -ethernetType   ethernetII
protocol config -enable802dot1qTag vlanStacked

stackedVlan setDefault

# Top (outer) VLAN element
vlan setDefault
vlan config -vlanID           2
vlan config -userPriority      1
vlan config -cfi                resetCFI
vlan config -mode              vIncrement
vlan config -repeat            10
vlan config -protocolTagId     vlanProtocolTag9200

# Top element must be modified by a setVlan
if { [stackedVlan setVlan 1] } {
    ixPuts $::ixErrorInfo
    return 1
}

```

```
# Next (inner) VLAN element
vlan setDefault
vlan config -vlanID          20
vlan config -userPriority    1
vlan config -cfi             resetCFI
vlan config -mode            vNestedIncrement
vlan config -repeat          100
vlan config -protocolTagId   vlanProtocolTag 9200

# Second element must be modified by a setVlan
if {[stackedVlan setVlan 2]} {
    ixPuts $::ixErrorInfo
    return 1
}

# Third stack element
vlan setDefault
vlan config -vlanID          42
vlan config -userPriority    2
vlan config -cfi             resetCFI
vlan config -protocolTagId   vlanProtocolTag 9100

# Third element must be added by addVlan
if {[stackedVlan addVlan]} {
    ixPuts $::ixErrorInfo
    return 1
}

if {[stackedVlan set $chas $card $port]} {
    ixPuts $::ixErrorInfo
    return 1
}

if {[stream set $chas $card $port 2]} {
    ixPuts $::ixErrorInfo
    return 1
}

ixWriteConfigToHardware portList
```

SEE ALSO

stream, protocol

NAME - vsrError

vsrError - configure the vsrError parameters for a port on a card on a chassis

SYNOPSIS	vsrError <i>sub-command options</i>
-----------------	-------------------------------------

DESCRIPTION	The vsrError command is used to insert deliberate errors in VSR equipped 10Gigabit Ethernet cards.
--------------------	---

STANDARD OPTIONS

General Control Options

enableChannelSwap <i>true / false</i>	If <i>true</i> , enables Channel Swapping. Channels 1-6 are swapped with Channels 7-12, to check for cable crossover. (<i>default = false</i>)
enableDelimiterInsert <i>true / false</i>	If <i>true</i> , enables the insertion of frame delimiters. Frame Delimiter Error Checking is also enabled. (<i>default = true</i>)
enableErrorCorrection <i>true / false</i>	If <i>true</i> , enables CRC error correction. (<i>default = true</i>)
enableProtectSwitch <i>true / false</i>	If <i>true</i> , enables the use of Protection Switching. Protection Switching is triggered when there is loss of synchronization on a single data channel. The data channel can be reconstructed, based on information in the Protection Channel and the other 9 data channels. This is a feature which is always present in the transmission, but which is optionally enabled by the receiver. (<i>default = true</i>)

Section BIP Error Insertion

bipErrorFrameCount	Specifies the number of consecutive frames, within a block of 256 frames, into which Section BIP Errors is injected. The errors repeats every 256 frames. (<i>default = 0</i>)
---------------------------	--

bipErrorMask	A one-byte mask which indicates which bits in the Section BIP B1 byte to XOR to generate the error. (<i>default = 0</i>)
---------------------	--

bipInsertionMode	The mode in which errors are inserted. Options include:
-------------------------	---

Option	Value	Usage
<i>vsrErrorInsertNone</i>	0	(<i>default</i>) Don't insert any errors.
<i>vsrErrorInsertContinuously</i>	1	Insert errors continuously, until <i>stop</i> is called.
<i>vsrErrorMomentarily</i>	2	Insert errors once.

CRC Error Insertion

crcChannelSelection	Selects which channels to insert errors into. A 12-bit bitmask is used to indicate the channels. A '1' indicates that errors should be inserted. Channel 1 is the least significant bit. The values :: <i>vsrChannel1</i> through :: <i>vsrChannel7</i> can be or'd together to construct a channel mask. (<i>default = 0</i>)
----------------------------	--

crcErrorBlockCount

The number of consecutive virtual blocks to inject CRC errors into, within a group of 16 virtual blocks. The errors are repeated every 16 blocks. (*default = 0*)

crcInsertionMode

The mode in which errors are inserted. Options include:

Option	Value	Usage
vsrErrorInsertNone	0	(<i>default</i>) Don't insert any errors.
vsrErrorInsertContinuously	1	Insert errors continuously, until <i>stop</i> is called.
vsrErrorMomentarily	2	Insert errors once.

Frame Delimiter Error Insertion**enableControlByte1**
true / false

If *true*, then the inserted value in *frameDelimiterControlByte1* is inserted as a control character. (*default = false*)

enableControlByte2Ch1To6
true / false

If *true*, then the inserted value in *frameDelimiterControlByte2Ch1To6* is inserted as a control character. (*default = false*)

enableControlByte2Ch7To12
true / false

If *true*, then the inserted value in *frameDelimiterControlByte2Ch7To12* is inserted as a control character. (*default = false*)

enableControlByte3
true / false

If *true*, then the inserted value in *frameDelimiterControlByte3* is inserted as a control character. (*default = false*)

frameDelimiter
ChannelSelection

Selects which channels to insert errors into. A 12-bit bitmask is used to indicate the channels. A '1' indicates that errors should be inserted. Channel 1 is the least significant bit. The values ::vsrChannel1 through ::vsrChannel7 can be or'd together to construct a channel mask. (*default = 0*)

frameDelimiterControl
Byte1

For the first delimiter byte, the 8b injected value. The default value (hex BC) translates to Codeword K28.5. (*default = 0xBC*)

frameDelimiterControl
Byte2Ch1To6

For the second delimiter byte, the 8b injected value in channels 1-6. The default value (hex 23) translates to Codeword D3.1. The delimiter for Channels 1-6 is different from that used for Channels 7-12, so the polarity of the patchcord / channel order can be detected. (*default = 0x23*)

frameDelimiterControl
Byte2Ch7To12

For the second delimiter byte, the 8b injected value in channels 7-12. The default value shown (hex 55) translates to Codeword D21.2. The delimiter for Channels 0-5 is different from that used for Channels 7-12, so the polarity of the patchcord/ channel order can be detected. (*default = 0x55*)

frameDelimiterControl
Byte3

For the third delimiter byte, the 8b injected value. The default value (hex BC) translates to Codeword K28.5. (*default = 0xBC*)

frameDelimiterError
FrameCount

The number of consecutive frames to inject CRC errors into, within a block of 16 frames. The error is repeated for each block of 16 frames. If the count = 0, frame delimiter error injection is disabled. (*default = 0*)

frameDelimiterError
Mask

A 3-bit mask of where errors is inserted. The '1' bit corresponds to the B1 byte, the '2' bit corresponds to the B2 byte and the '4' bit corresponds to the B3 byte. For example, a value of '5' inserts errors into the B1 and B3 bytes. (*default = 0*)

frameDelimiter
InsertionMode

The mode in which errors are inserted. Options include:

Option	Value	Usage
<code>vsrErrorInsertNone</code>	0	(default) Don't insert any errors.
<code>vsrErrorInsertContinuously</code>	1	Insert errors continuously, until <code>stop</code> is called.
<code>vsrErrorMomentarily</code>	2	Insert errors once.

Channel Skew Error Insertion

channelSkew
ChannelSelection

Selects which channels to insert errors into. A 12-bit bitmask is used to indicate the channels. A '1' indicates that errors should be inserted. Channel 1 is the least significant bit. The values `::vsrChannel1` through `::vsrChannel7` can be or'd together to construct a channel mask. (default = 0)

channelSkew
DelayTime

The number of clock cycles of delay to be applied to the selected channels. (default = 1)

channelSkew
InsertionMode

The mode in which errors is inserted. Options include:

Option	Value	Usage
<code>vsrErrorInsertNone</code>	0	(default) Don't insert any errors.
<code>vsrErrorInsertContinuously</code>	1	Insert errors continuously, until <code>stop</code> is called.
<code>vsrErrorMomentarily</code>	2	Insert errors once.

channelSkewMode

The skew injection mode. Options include:

Option	Value	Usage
<code>vsrErrorSingleChannelMode</code>	0	(default) Only delay a single channel.
<code>vsrErrorMultiChannelMode</code>	1	Each of the selected channels is delayed.

8b/10b Code Word Error Insertion

enableControl
CharCodeWord
true / false

If `true`, the injected code word is a control character. (default = false)

enableDisparity
ErrorCodeWord
true / false

If `true` disparity errors are injected. Note: disparity errors may cause codeword violations. (default = false)

error8b10bChannel
Selection

Selects which channels to insert errors into. A 12-bit bitmask is used to indicate the channels. A '1' indicates that errors should be inserted. Channel 1 is the least significant bit. The values `::vsrChannel1` through `::vsrChannel7` can be or'd together to construct a channel mask. (default = 0)

error8b10bCodeWord
Count

Specifies the number of consecutive codewords, per block of 16 code words, into which code violations are injected. This pattern is repeated for every block of 16 codewords. (default = 0)

error8b10bCodeWord
Value

Specifies the 8b value for the code word to be injected. (default = 0)

**error8b10bInsertion
Mode**

The mode in which errors are inserted. Options include:

Option	Value	Usage
vsrErrorInsertNone	0	(default) Don't insert any errors.
vsrErrorInsertContinuously	1	Insert errors continuously, until <i>stop</i> is called.
vsrErrorMomentarily	2	Insert errors once.

COMMANDS

The **vsrError** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

vsrError cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **vsrError** command.

vsrError config *option value*

Modify the **vsrError** configuration options of the port. If no *option* is specified, returns a list describing all of the available vsrError options (see STANDARD OPTIONS) for port.

vsrError get *chasID cardID portID*

Gets the current VSR error configuration of the port with id *portID* on card *cardID*, chassis *chasID*. Call this command before calling **vsrError cget *option*** to get the value of the configuration option.

vsrError insertError *vsrErrorType chasID cardID portID*

Insert a single instance of the error indicated in *vsrErrorType* into the indicated port. The choices of *vsrErrorType* are:

Option	Value	Usage
vsrErrorSectionBip	1	Section BIP errors
vsrErrorCrc	2	CRC errors
vsrErrorFrameDelimiter	3	Frame delimiter errors
vsrErrorChannelSkew	4	Channel skew errors
vsrError8b10bCode	5	8b/10b code word errors
vsrErrorAll	0xEF	All errors
vsrErrorStopAll	0xFF	Stop all errors

vsrError set *chasID cardID portID*

Sets the **vsrError** configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **vsrError config *option value*** command.

vsrError setDefault

Sets to IxTclHal default values for all configuration options.

vsrError start chasID cardID portID

Insert errors as indicated by the various options into the indicated port. **vsrError stop** should be used to stop error insertions if any of the **InsertionMode*'s are set to *vsrErrorInsertContinuously*.

vsrError stop chasID cardID portID

Stops all errors insertion on the indicated port.

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that a VSR card is in slot 39
set card 39
set portList [list [list $chas $card 1]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# see if the card supports VSR
if {[port isValidFeature portFeatureVsr $chas $card 1] == 0} {
    ixPuts "Card $card is not an VSR card"
    return 1
}

# ... Normal port, protocol, stream operations

# Insert channel skew on channels 1, 3 and 5 continuously
vsrError setDefault
vsrError config -channelSkewMode
vsrErrorMultiChannelMode
vsrError config -channelSkewChannelSelection 21
```

```
vsrError config -channelSkewInsertionMode
vsrErrorInsertContinuously
vsrError config -channelSkewDelayTime      1
if [vsrError set $chas $card 1] {
    ixPuts "Can't vsrError set $chas:$card:1"
    return 1
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*vsrStat*

NAME - vsrStat

vsrStat - read 10Gigabit VSR statistics

SYNOPSIS	vsrStat <i>sub-command options</i>
DESCRIPTION	The vsrStat command is used to read global and per channel VSR statistics for 10Gigabit Ethernet cards.
STANDARD OPTIONS	
Global Statistics	
rxChannelProtection Disabled	<i>Read-Only</i> . <i>True</i> or <i>false</i> , indicating the status of the channel protection on the receiving interface.
rxChannelSkewError	<i>Read-Only</i> . <i>True</i> or <i>false</i> , indicating the status of the channel skew error detection on the receiving interface.
rxChannelSkewFirst	<i>Read-Only</i> . Indicates the channel number of the earliest channel to arrive on the receiving interface. If more than one channel arrives at the same time, Channel #1 has the highest priority and so on.
rxChannelSkewLast	<i>Read-Only</i> . Indicates the channel number of the latest channel to arrive on the receiving interface. If more than one channel arrives at the same time, Channel #1 has the highest priority, and so on.
rxChannelSkewMax	<i>Read-Only</i> . This counter increments every time the channel skew is equal to or greater than the maximum channel skew.
rxChannelSwapped	<i>Read-Only</i> . <i>True</i> indicates one or more channel swap errors and <i>false</i> indicates no errors.
rxCodeWordViolation Error	<i>Read-Only</i> . <i>True</i> indicates one or more 8b/10b code word violation errors and <i>false</i> indicates no errors.
rxCrcCorrectedError Counter	<i>Read-Only</i> . The number of corrected CRC block errors accumulated on the receiving interface.
rxCrcCorrection Disabled	<i>Read-Only</i> . <i>True</i> or <i>false</i> , indicating the status of the CRC correction on the receiving interface.
rxCrcError	<i>Read-Only</i> . <i>True</i> indicates one or more detected CRC errors and <i>false</i> indicates no errors.
rxCrcUnCorrectedError Counter	<i>Read-Only</i> . The number of uncorrected CRC block errors accumulated on the receiving interface.
rxHardwareError	<i>Read-Only</i> . The number of hardware errors detected on the receive side.

rxLossOfSynchronizationCounter	<i>Read-Only.</i> Indicates the number of times that a protection channels was in the loss of synchronization state.
rxMultiLossOfSynchronizationCounter	<i>Read-Only.</i> Indicates the number of times that two or more data or protection channels were in the Loss of Synchronization state.
rxMultiLossOfSynchronizationStatus	<i>Read-Only.</i> <i>True</i> indicates that two or more data or protection channels are in the Loss of Synchronization state.
rxOutOfFrameCounter	<i>Read-Only.</i> Indicates the number of frame errors for the receiving interface.
rxOutOfFrameStatus	<i>Read-Only.</i> <i>True</i> indicates one or more out of frame errors for the receiving interface and <i>false</i> indicates no errors.
rxSectionBipErrorCounter	<i>Read-Only.</i> The number of Section BIP errors detected on the receiving interface.
txHardwareError	<i>Read-Only.</i> The number of hardware errors detected on the transmit side.
txOutOfFrameCounter	<i>Read-Only.</i> The number of out of frame errors detected on the transmit side.
txOutOfFrameStatus	<i>Read-Only.</i> <i>True</i> indicates one or more out of frame errors for the transmit interface and <i>false</i> indicates no errors.
txSectionBipErrorCounter	<i>Read-Only.</i> The number of Section Bit Interleaved Parity (BIP) errors which have been detected on the transmit interface.

Per-Channel Statistics

rxCodeWordViolationCounter	<i>Read-Only.</i> This per-channel statistic indicates the number of codeword violations detected on the receiving channel interface. Codeword violations include running disparity errors, undefined codewords, and any control characters besides K28.5.
rxCrcErrorCounter	<i>Read-Only.</i> This per-channel statistic indicates the number of corrected and uncorrected errors on the receive interface.
rxLossOfSynchronization	<i>Read-Only.</i> This per-channel statistic indicates the loss of synchronization status of the receiving interface as a <i>true</i> or <i>false</i> value.
rxOutOfFrame	<i>Read-Only.</i> This per-channel statistic indicates the out of frame status of the receiving interface for a particular channel as a <i>true</i> or <i>false</i> value.

COMMANDS

The **vsrStat** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

vsrStat cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **vsrStat** command.

vsrStat get chasID cardID portID

Gets the current VSR statistics of the indicated. Call this command before calling **vsrStat cget option** to get the value of the global statistics. Also call **vsrStat getChannel channelID** before getting statistics for a particular channel.

vsrStat getChannel *channelID*

Gets the statistics for the channel indicated by *channelID*, which must be a value between 1 and 12. The per-channel statistics are then available through the use of **vsrStat cget option**.

vsrStat set *chassisID cardID portID*

Sets the vsrStat configuration of the port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **vsrStat config option value** command.

EXAMPLES

```
package require IxTclHal

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that a VSR card is in slot 39
set card 39
set portList [list [list $chas $card 1]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# see if the card supports VSR
if {[port isValidFeature portFeatureVsr $chas $card 1] == 0} {
    ixPuts "Card $card is not a VSR card"
    return 1
}

# ... Normal port, protocol, stream operations

ixWriteConfigToHardware portList

# Let's get some global stat for port 1
if [vsrStat get $chas $card 1] {
    ixPuts "Can't vsrStat get for $chas.$card.1"
```

```
        return 1
    }

    if {[vsrStat cget -rxChannelSkewError]} {
        ixPuts "Channel error: "
        set first [vsrStat cget -rxChannelSkewFirst]
        set latest [vsrStat cget -rxChannelSkewLast]
        ixPuts " Channels $first - $latest"

        # And now some stat for the earliest skewed channel
        if [vsrStat getChannel $first] {
            ixPuts "Can't vsrStat getChannel $first"
            return 1
        }
        ixPuts -nonewline "Number of Crc Errors on channel $first: "
        ixPuts [vsrStat cget -rxCrcErrorCounter]
    }

    # Let go of the ports that we reserved
    ixClearOwnership $portList
    # Disconnect from the chassis we're using
    ixDisconnectFromChassis $host
    # If we're running on UNIX, disconnect from the TCL Server
    if [isUNIX] {
        ixDisconnectTclServer $host
    }
}
```

SEE ALSO[vsrError](#)

NAME - weightedRandomFramesize

weightedRandomFramesize - configure weighted random frame sizes

SYNOPSIS	weightedRandomFramesize <i>sub-command options</i>
DESCRIPTION	<p>The <i>weightedRandomFramesize</i> command is used to configure possible different modes of generating random frame sizes for a particular stream. This command is used for ports which support this feature and where the port has been programmed for random stream generation with:</p> <pre><i>stream</i> config -frameSizeType sizeRandom</pre> <p>The availability of this feature on a given port may be tested with the <i>portisValidFeature... portFeatureRandomFrameSizeWeightedPair</i>.</p> <p>Four basic types of random streams are available, and are set in the <i>randomType</i> option:</p> <ul style="list-style-type: none"> • Uniform: identical to previous implementations of the random framesize feature. A uniform set of random values between a minimum and maximum value are generated. The min/max values are set in the <i>stream</i> command's <i>frameSizeMIN</i> and <i>frameSizeMAX</i> options. • Pre-programmed: a number of pre-programmed distributions are available, corresponding to standard traffic models found in various applications. See the <i>randomType</i> option description below. • Custom: a distribution may be custom programmed for a stream. Pairs of frame size-weights are added to a list. Frame sizes may be any value valid for the port. Weights may be any value, such that the total of all of the weights is less than 2048. Pairs are added to the list using the <i>addPair</i> sub-command. • Gaussian: up to four gaussian curves may be summed up to generate a random distribution. Each curve is specified in the <i>center</i>, and <i>widthAtHalf</i> options and set by the <i>updateQuadGaussianCurve</i> sub-command. The <i>weight</i> option controls the distribution of values among the four curves. <p>For the pre-programmed and custom choices, the weights for all of the frame sizes are added up. Each frame size is then given a proportion of the total number of frames as dictated by its weight value. For example, one of the pre-programmed distributions is (64:7, 594:4, 1518:1). In this case, the total of the weights is 12 (7+4+1). Frames are randomly generated such that 64-byte frames are 7/12 of the total, 594-byte frames are 4/12 of the total and 1518-byte frames are 1/12 of the total.</p> <p>Note that <i>stream</i> get must be called before this command's get sub-command.</p>
<hr/>	
STANDARD OPTIONS	
center	If <i>randomType</i> is set to <i>randomQuadGaussian</i> , then this is used to indicate the center value of the curve, expressed in framesize. Fractional values are permitted. This and the <i>weight</i> and <i>widthAtHalf</i> options are associated with one of the four

available curves by the *updateQuadGaussianCurve* sub-command. (*default = 200.0*)

pairList

Read-only. After a *set* operation, this option holds a TCL list with the frame size-weight pairs.

randomType

The type of random weighted frames sizes to be generated.

Option	Value	Usage
<i>randomUniform</i>	0	(<i>default</i>) A uniform distribution between the min/max values found in the <i>frameSizeMIN</i> and <i>frameSizeMAX</i> options in the <i>stream</i> command.
<i>randomWeightedPair</i>	1	Custom weighted pairs are used for the distribution. Pairs are added to the list using the <i>addPair</i> sub-command. The sum of all of the weights must be less than 2048.
<i>randomQuadGaussian</i>	3	Up to four gaussian curves may be specified in the <i>center</i> , <i>weight</i> and <i>widthAtHalf</i> options.
<i>randomCisco</i>	4	A pre-programmed distribution is used: 64:7, 594:4 and 1518:1.
<i>randomIMIX</i>	5	A pre-programmed distribution is used: 64:7, 570:4 and 1518:1.
<i>randomTolly</i>	7	A pre-programmed distribution is used: 64:55, 78:5, 576:17, and 1518: 23.
<i>randomRPRTrimodal</i>	8	A pre-programmed distribution is used: 64:60, 512:20, and 1518:20.
<i>randomRPRQuadmodal</i>	9	A pre-programmed distribution is used: 64:60, 512:20, 1518:20 and 9000:20.

weight

If *randomType* is set to *randomQuadGaussian*, then this is used to indicate the relative weight of the values from this curve with respect to the other three curves. This and the *widthAtHalf* and *center* options are associated with one of the four available curves by the *updateQuadGaussianCurve* sub-command. (*default = 1*)

widthAtHalf

If *randomType* is set to *randomQuadGaussian*, then this is used to indicate the width of the curve at its half-value height, expressed in framesize. Fractional values are permitted. This and the *weight* and *center* options are associated with one of the four available curves by the *updateQuadGaussianCurve* sub-command. The valid range is .01 to 30000. (*default = 100.0*)

COMMANDS

The **weightedRandomFramesize** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

weightedRandomFramesize addPair framesize weight

Adds the *framesize-weight* pair to the *pairList*. Multiple pairs which use the same *framesize* have their weights effectively added together. Specific errors include:

- The value of *randomType* is not *randomWeightedPair*
- Memory exceeded

weightedRandomFramesize cget *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **weightedRandomFramesize** command.

weightedRandomFramesize config *option value*

Modify the weightedRandomFramesize configuration options of the port. If no *option* is specified, returns a list describing all of the available weightedRandomFramesize options (see STANDARD OPTIONS) for port.

weightedRandomFramesize delPair *framesize weight*

Deletes the first *framesize-weight* pair in the *pairList*. Specific errors include:

- The value of *randomType* is not *randomWeightedPair*
- The pair could not be found

weightedRandomFramesize get *chassisID cardID portID*

Gets the current configuration of the port with id *portID* on card *cardID*, chassis *chassisID*. Note that *stream* get must be called before this command's get sub-command. Call this command before calling **weightedRandomFramesize cget option** to get the value of the configuration option.

weightedRandomFramesize retrieveQuadGaussianCurve *curveId*

Retrieves the values associated with the Gaussian curve specified in *curveId* and sets them into the *center*, *widthAtHalf* and *weight* options of this command.

weightedRandomFramesize set *chassisID cardID portID*

Sets the weightedRandomFramesize configuration of the port with id *portID* on card *cardID*, chassis *chassisID* by reading the configuration option values set by the **weightedRandomFramesize config option value** command.

weightedRandomFramesizesetDefault

Sets to IxTclHal default values for all configuration options.

weightedRandomFramesize updateQuadGaussianCurve *curveId*

Sets the values associated with the Gaussian curve specified in *curveId* using the values in the *center*, *widthAtHalf* and *weight* options of this command.

EXAMPLES

```
package req IxTclHal

set hostname loopback
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
```

```

if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set ch [ixGetChassisID $host]

set cd 22
set prt 1
set portList [list [list $ch $cd $prt]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

stream config -frameSizeType      sizeRandom

weightedRandomFramesize setDefault
weightedRandomFramesize config -randomType randomWeightedPair
weightedRandomFramesize addPair 100   5
weightedRandomFramesize addPair 200   10

if [weightedRandomFramesize set $ch $cd $prt] {
    errorMsg " Error setting weighted 1 on port $ch $cd $prt "
    return $ ::TCL_ERROR
}

if [stream set $ch $cd $prt 1] {
    errorMsg " Error setting stream 1 on port $ch $cd $prt "
    return $ ::TCL_ERROR
}
# weightedRandomFramesize cget -pairList will return
# "{100 5} {200 10}"

weightedRandomFramesize setDefault
weightedRandomFramesize config -randomType randomUniform

if [weightedRandomFramesize set $ch $cd $prt] {
    errorMsg " Error setting weighted 1 on port $ch $cd $prt "
    return $ ::TCL_ERROR
}

stream config -frameSizeType      sizeRandom
stream config -frameSizeMIN      100
stream config -frameSizeMAX      1000

if [stream set $ch $cd $prt 2] {
    errorMsg " Error setting stream 2 on port $ch $cd $prt "
    return $ ::TCL_ERROR
}

weightedRandomFramesize setDefault
weightedRandomFramesize config -randomType randomCisco

if [weightedRandomFramesize set $ch $cd $prt] {
    errorMsg " Error setting weighted 1 on port $ch $cd $prt "
}

```

```

        return $ ::TCL_ERROR
    }

    if [stream set $ch $cd $prt  3] {
        errorMsg " Error setting stream on port $ch $cd $prt "
        return $::TCL_ERROR
    }
# weightedRandomFramesize cget -pairList will return
# "{ 64 7 } { 594 4 } { 1518 1 }"

weightedRandomFramesize setDefault
weightedRandomFramesize config -randomType      randomQuadGaussian
weightedRandomFramesize config -center          256.0
weightedRandomFramesize config -widthAtHalf    128.0
weightedRandomFramesize config -weight         1
if [weightedRandomFramesize updateQuadGaussianCurve 1] {
    ixPuts $::ixErrorInfo
    return $ ::TCL_ERROR
}
weightedRandomFramesize config -center          512.0
weightedRandomFramesize config -widthAtHalf    256.0
weightedRandomFramesize config -weight         4
if [weightedRandomFramesize updateQuadGaussianCurve 2] {
    ixPuts $::ixErrorInfo
    return $ ::TCL_ERROR
}
weightedRandomFramesize config -center          1024.0
weightedRandomFramesize config -widthAtHalf   450.0
weightedRandomFramesize config -weight         8
if [weightedRandomFramesize updateQuadGaussianCurve 3] {
    ixPuts $::ixErrorInfo
    return $ ::TCL_ERROR
}
weightedRandomFramesize config -center          1500.0
weightedRandomFramesize config -widthAtHalf   12.0
weightedRandomFramesize config -weight         1
if [weightedRandomFramesize updateQuadGaussianCurve 4] {
    ixPuts $::ixErrorInfo
    return $ ::TCL_ERROR
}
if [weightedRandomFramesize set $ch $cd $prt] {
    ixPuts $::ixErrorInfo
    return $ ::TCL_ERROR
}
if [stream set $ch $cd $prt  4] {
    errorMsg " Error setting stream on port $ch $cd $prt "
    return $::TCL_ERROR
}

ixWriteConfigToHardware portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*stream*

NAME - xau

xau - XAUI power and clock settings

SYNOPSIS

xau *sub-command options*

DESCRIPTION

The **xau** command is used to change power and clock settings on 10Gigabit XAUI cards.

STANDARD OPTIONS

clockType Indicates whether the XAUI clock is internally or externally supplied. Options include:

Option	Value	Usage
<i>xauClockInternal</i>	0	(default) Timing is supplied by the internally generated clock.
<i>xauClockExternal</i>	1	Timing is supplied by an externally provided clock.

extraClockExternal1 (*default = 1*)

extraClockExternal2 (*default = 1*)

podPower true / false If *true*, the card applies 5V power limited to 500ma at pin 5 of the D15 MDIO connector on the front panel. (*default = 0*)

userPower true / false If *true*, the card applies 5V power limited to 500ma at pin 4 of the D15 MDIO connector on the front panel. (*default = 1*)

COMMANDS

The **xau** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

xau cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **xau** command.

xau get chasID cardID portID

Gets the current XAUI configuration of the indicated port. Call this command before calling **xau cget option**.

xau set chasID cardID portID

Sets the XAUI configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **xau config option value** command.

EXAMPLES

```
package require IxTclHal
```

```
# Connect to chassis and get chassis ID
set host localhost
```

```
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that a VSR card is in slot 59
set card 59
set portList [list [list $chas $card 1]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# see if the card supports XAUI
if {[port isValidFeature portFeatureXaui $chas $card 1] == 0} {
    ixPuts "Card $card is not a XAUI card"
    return 1
}

# Apply pod and user power to MDIO pins 5 and 4, respectively
xaui setDefault
xaui config -podPower true
xaui config -userPower true
if [xaui set $chas $card 1] {
    ixPuts "Can't xaui set $chas.$card.1"
    return 1
}

ixWritePortsToHardware      portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

NAME - **xfp**

xfp - UNIPHY-XFP settings

SYNOPSIS	xfp <i>sub-command options</i>
-----------------	---------------------------------------

DESCRIPTION	The xfp command is used to change monitor settings for UNIPHY-XFP cards.
--------------------	---

STANDARD OPTIONS

enableMonitorLos true / false	If <i>true</i> , enables the port to monitor Loss of Signal. In this case, the Loss of Signal status is used to determine Link State. (<i>default = true</i>)
enableMonitorModule ReadySignal true / false	If <i>true</i> , enables the port to monitor whether the module is ready. In this case, no transmit, received or statistics operations are performed until the module is ready. (<i>default = true</i>)

COMMANDS	The xfp command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.
-----------------	--

xfp cget option

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **xfp** command.

xfp get chasID cardID portID

Gets the current xfp configuration of the indicated port. Call this command before calling **xfp cget option**.

xfp set chasID cardID portID

Sets the xfp configuration of the port with id *portID* on card *cardID*, chassis *chasID* by reading the configuration option values set by the **xfp config option value** command.

EXAMPLES	package require IxTclHal
-----------------	--------------------------

```

# Connect to chassis and get chassis ID
set host localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

```

```
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Assuming that a VSR card is in slot 59
set card 59
set portList [list [list $chas $card 1]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# see if the card supports XAUI
if {[port isValidFeature portFeatureXaui $chas $card 1] == 0} {
    ixPuts "Card $card is not a XAUI card"
    return 1
}

# Disable both monitor settings
xfp config -enableMonitorLos           false
xfp config -enableMonitorModuleReadySignal false
if [xfp set $chas $card 1] {
    ixPuts "Can't xfp set $chas.$card.1"
    return 1
}

ixWritePortsToHardware      portList

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

B

Utility Commands

NAME - byte2IpAddr

byte2IpAddr - convert 4 hex bytes into an IP address in dotted notation

SYNOPSIS

byte2IpAddr <hexVal>

DESCRIPTION

The **byte2IpAddr** command converts 4 hex bytes into an IP address in dotted notation. It can be used in scripts where IP addresses are read from the capture buffer in hexadecimal format, for example.

EXAMPLE

byte2IpAddr "C0 02 0A 0C"

Returns 192.2.10.12

SEE ALSO

dectohex, hextodec, host2addr

NAME - calculateFPS

calculateFPS - calculates the frame rate, in frames/second

Note: This command has been deprecated. Use [calculateMaxRate](#) instead.

SYNOPSIS

`calculateFrameRate chassis card port [percentMaxRate frameSize preambleOrAtmEncap]`

DESCRIPTION

The **calculateFPS** command calculates the frame rate for a particular port type based on the percentage of the maximum rate, frame size and the preamble size.

COMMAND

The **calculateFPS** command is invoked with the following arguments.

`calculateFPS chassis card port [percentMaxRate framesize preambleSize]`

where:

chassis, card, port: A port of the type that you wish the frame rate calculated for

percentMaxRate: The percentage of the maximum rate (*default = 100*)

frameSize: the size of the frame (*default = 64*)

preambleOrAtmEncap: The size of the preamble, or the ATM encapsulation used for ATM cards. The values for ATM encapsulation may be found in the *encapsulation* option of the [atmHeader](#) command. (*default = 8*)

EXAMPLE

`calculateFPS 1 1 1 80 64 8`

Returns 11904.7619048

SEE ALSO

[calculateMaxRate](#), [calculatePercentMaxRate](#)

NAME - **calculateGapBytes**

calculateGapBytes - calculates the inter-frame gap for a port, expressed in equivalent number of data bytes.

Note: this command has been deprecated. Use *calculatePercentMaxRate* instead.

SYNOPSIS	<i>calculateGapBytes chassis card port frameRate frameSize preambleSize</i>
-----------------	---

DESCRIPTION	The calculateGapBytes command calculates the IFG in terms of the number of data bytes that could fit in the gap, based on the frame rate, frame size and preamble size.
--------------------	--

COMMAND	The calculateGapBytes command is invoked with the following arguments.
----------------	---

*calculateGapBytes chassis card port frameRate
[frameSize preambleSize]*

where:

chassis, card, port: A port of the type that you wish the gap calculated for

frameRate: The input frame rate in FPS

frameSize: The size of the frame (*default = 64*)

preambleSize: The size of the preamble (*default = 8*)

EXAMPLE	<i>calculateGapBytes 1 1 1 1000</i>
----------------	-------------------------------------

Returns 1178

SEE ALSO	<i>calculateMaxRate, calculatePercentMaxRate</i>
-----------------	--

NAME - calculateMaxRate

calculateMaxRate - calculates the inter-frame gap for a port

SYNOPSIS

`calculateMaxRate chassis card port frameSize preambleOrAtmEncap`

DESCRIPTION

The **calculateMaxRate** command calculates the maximum frame rate for a port, based on the frame size and preamble size.

COMMAND

The **calculateMaxRate** command is invoked with the following arguments.

`calculateMaxRate chassis card port [framesize preambleSize]`

where:

chassis, card, port: A port of the type that you wish the maximum frame rate calculated for

frameSize: The size of the frame (*default* = 64)

preambleOrAtmEncap: The size of the preamble, or the ATM encapsulation used for ATM cards. The values for ATM encapsulation may be found in the *encapsulation* option of the [*atmHeader*](#) command. (*default* = 8)

EXAMPLE

`calculateMaxRate 1 1 1 1518`

Returns 813

SEE ALSO

[*calculatePercentMaxRate*](#)

NAME - calculatePercentMaxRate

calculatePercentMaxRate - calculates what percentage of the maximum rate a particular frame rate is

SYNOPSIS

calculatePercentMaxRate *chassis card port frameRate [frameSize preambleOrAtmEncap]*

DESCRIPTION

The **calculatePercentMaxRate** command calculates what percentage of the maximum rate a particular frame rate is for a particular port type based on the frame size and the preamble size.

COMMAND

The **calculatePercentMaxRate** command is invoked with the following arguments.

```
calculatePercentMaxRate chassis card port frameRate
[frameSize preambleSize]
```

where:

chassis, card, port: A port of the type that you wish the frame rate calculated for

frameRate: The input frame rate in FPS.

frameSize: The size of the frame (*default = 64*)

preambleOrAtmEncap: The size of the preamble, or the ATM encapsulation used for ATM cards. The values for ATM encapsulation may be found in the *encapsulation* option of the *atmHeader* command. (*default = 8*)

EXAMPLE

```
package require IxTclHal

# In this example, we'll find all the 10/100/1000 cards
# and program their first port to 128,000 FPS for 64 byte packets
# and 8 byte preamble

set host localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```

}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis' number of cards
chassis getFromID $chas
set ncards [chassis cget -maxCardCount]
ixPuts "Chassis $chas, $ncards cards"

for {set i 1} {$i <= $ncards} {incr i} {
    # Check for missing card
    if {[card get $chas $i] != 0} {
        continue
    }
    set typeName      [card cget -typeName]

    # If the card is a 10/100 RMII, play with its settable
    # parameters
    if {[string first "1000" $typeName] != -1} {
        ixPuts "Card $i: $typeName"
        set portList [list [list $chas $i 1]]
        if [ixTakeOwnership $portList] {
            ixPuts $::ixErrorInfo
            return 1
        }
        port setFactoryDefaults $chas $i 1
        port config -speed 1000
        port set $chas $i 1

        set percentMax [calculatePercentMaxRate $chas $i 1 128000
64 8]
        stream config -rateMode usePercentRate
        stream config -percentPacketRate $percentMax
    }

#####
#
# NOTE: in the past, this was done with the CalculateGap
#       command. For example:
#
#       set gapTicks [calculateGap 128000 64 8 $card $i 1]
#       stream config -rateMode useGap
#       stream config -gapUnit    gapClockTicks
#       stream config -ifg       $gapTicks
#
# This no longer works for new Ixia cards, since the
definition
# of a clock tick varies per board. calculatePercentMaxRate
# is card independent and works in all cases
#
#####

stream setDefault
stream config -framesize          64
stream config -preambleSize      8
if [stream set $chas $i 1 1] {

```

```
        ixPuts $ixErrorInfo
        return 1
    }
    ixWriteConfigToHardware portList
}
}

# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*calculateMaxRate*

NAME - .cleanUp

cleanUp - end a test and cleanup all variables

SYNOPSIS

cleanUp

DESCRIPTION

The **cleanUp** command reliably terminates a test and resets all important parameters. This includes

- Removing all chassis from the chassis chain
- Disconnects from a TCL Server, if necessary
- Removes the effect of a *package require IxTclHal*
- Resets all commands back to their default state
- Closes all open files

EXAMPLE

cleanUp

SEE ALSO

NAME - clearAllMyOwnership

clearAllMyOwnership - clear all current port ownership

SYNOPSIS clearAllMyOwnership

DESCRIPTION The **clearAllMyOwnership** command releases all port ownership for the currently logged on user.

EXAMPLE clearAllMyOwnership

SEE ALSO *ixClearOwnership*

NAME - dectohex

dectohex - convert a decimal number to a hexadecimal number

SYNOPSIS

dectohex <decimal number>

DESCRIPTION

The **dectohex** command converts a decimal number to a hexadecimal number.

EXAMPLE

dectohex 10

Returns A

SEE ALSO

hextodec, host2addr, byte2IpAddr

NAME - disableUdfs

disableUdfs - disables all UDFs in the argument list

SYNOPSIS disableUdfs *udfIDlist*

DESCRIPTION The **disableUdfs** command cycles through all the UDF numbers in the list argument list and disables them.

COMMAND The **disableUdfs** command is invoked with the following arguments.
 disableUdfs *udfList*
 where *udfList* is a list of UDF numbers 1 to 4.

EXAMPLE disableUdfs {1 3}

SEE ALSO *udf, stream*

NAME - enableEvents

enableEvents - log errors and warnings to a log file

SYNOPSIS	enableEvents { <i>true</i> <i>false</i> }
DESCRIPTION	The enableEvents command enables or disables the creation of a log file in the <i>C:\Program Files\Ixia</i> folder. The log file is named with the creation date and time. This value is <i>true</i> by default for Windows operating systems and <i>false</i> by default for Unix systems.
COMMAND	The enableEvents command is invoked with the following arguments. <code>enableEvents <i>true</i></code>
SEE ALSO	

NAME - errorMsg

errorMsg - logs text to the error file

SYNOPSIS **errorMsg [-nonewline] arg...**

DESCRIPTION The **errorMsg** command outputs its arguments to the error file with or without a new line.

ARGUMENTS

-nonewline If present, suppresses a newline at the end of the output

arg ... Arguments which are concatenated together and written to the error file.

RETURNS

0 No error; the command was successfully delivered to the IxServer

1 Error; the command was delivered to the IxServer but it could not process the message

EXAMPLE **errorMsg -nonewline "This will write to the errorFile"**

SEE ALSO [*getErrorMessage*](#)

NAME - getErrorString

getErrorString - return an error string corresponding to an error number

SYNOPSIS `getErrorString <error number>`

DESCRIPTION The **getErrorString** command converts an error number to a text string.

EXAMPLE

```
% ixutils getErrorString 1  
General Error. Check method parameters.  
% ixutils getErrorString 2  
Version mismatch between IxServer and Tcl client.
```

SEE ALSO

NAME - getStatLabel

getStatLabel - return a statistic value for a statistic.

SYNOPSIS getStatLabel <*string*>

DESCRIPTION The **getStatLabel** command gets the statistic value for a specified statistic.

EXAMPLE getStatLabel sArpInstalled

SEE ALSO

NAME - hextodec

hextodec - convert a hexadecimal number to a decimal number

SYNOPSIS `hextodec <hex number>`

DESCRIPTION The **hextodec** command converts a hexadecimal number to a decimal number.

EXAMPLE `hextodec 7a`

Returns 122

SEE ALSO *dectohex, host2addr, byte2IpAddr*

NAME - host2addr

host2addr - convert an IP address in dotted notation to a list of hex bytes

SYNOPSIS

host2addr <*IP address*>

DESCRIPTION

The **host2addr** command converts an IP address in dotted notation to a list of hex bytes. This command is useful in scripts where you specify an IP address in dotted notation and it needs to be converted into 4 hexadecimal byte format to store as a list.

EXAMPLE

host2addr 192.1.10.12

Returns C0 01 0A 0C

SEE ALSO

dectohex, host2addr, byte2IpAddr

NAME - **logMsg**

logMsg - logs text to the log file

SYNOPSIS

logMsg [-nonewline] arg...

DESCRIPTION

The **logMsg** command outputs its arguments to the log file with or without a new line.

ARGUMENTS

-nonewline If present, suppresses a newline at the end of the output

arg ... Arguments which are concatenated together and written to the log file.

RETURNS

0 No error; the command was successfully delivered to the IxServer

1 Error; the command was delivered to the IxServer but it could not process the message

EXAMPLE

`logMsg -nonewline "This will write to the logFile"`

SEE ALSO

logOn, logOff, ixPuts

NAME - logOff

logOn - disables logging.

SYNOPSIS logOff

DESCRIPTION The **logOff** command is used to turn off logging.

STANDARD OPTIONS

None

EXAMPLE logoff

SEE ALSO *ixProxyConnect, logOn*

NAME - **logOn**

logOn - enables logging.

SYNOPSIS

logOn *filename*

DESCRIPTION

The **logOn** command is used to turn on logging. The log file is configured with the command.

STANDARD OPTIONS

filename

The filename to log output under.

EXAMPLE

`logOn "c:/program files/ixia/log.log"`

SEE ALSO

ixProxyConnect, logOff

NAME - mpexpr

mpexpr - performs arbitrary precision arithmetic

SYNOPSIS	mpexpr <expression>
DESCRIPTION	mpexpr works much like Tcl's native expr , but does all calculations using an arbitrary precision math package. mpexpr numbers can be any number of digits, with any decimal precision. Final precision is controlled by a Tcl variable mp_precision , which can be any reasonable integer, limiting only the number of digits to the right of the decimal point.
COMMAND	The mpexpr command should be used on all 64-bit values as marked in the citations below.
EXAMPLE	<pre>package require Mpexpr set \$::mp_precision 25 set y 42 set eExpY [mpexpr exp(\$y)] puts [mpformat %f \$eExpY]</pre>
SEE ALSO	

NAME - showCmd

showCmd - show the current value of a TCL API command's values

SYNOPSIS showCmd <TCL API command>

DESCRIPTION **showCmd** is a very useful command that may be used to display the current value of a command's options. It may be typed into an interactive *wish* shell or included as a command in a TCL script.

COMMAND

showCmd *command*

command The name of any of the command.

EXAMPLE showCmd port
 showCmd rprFairness

SEE ALSO

NAME - user

user - configure the user related parameters

SYNOPSIS

user *sub-command options*

DESCRIPTION

The **user** command is used to configure user related information. This information is used when the RFC2544, RFC 2285 and non-RFC tests are executed and results are produced. It helps in the identification of the user and used for reference.

STANDARD OPTIONS

comments	A comment associated with the test.
productname	Name of the DUT being tested.
version	Version number of the product.
serial#	Serial number of the product.
username	The name of the user running the tests.

COMMAND

The **user** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

user *cget option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **user** command.

user *config option value*

Modify the configuration options of the user. If no *option* is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for user.

user *setDefault*

Sets default values for all configuration options.

EXAMPLES

```
package require IxTclHal

user setDefault
user config -comments "Special XYZ test"
user config -productname "Super router 2000"
user config -version "0.1"
user config -serial# "1"
user config -username "QA Specialist 14"

ixPuts [user cget -productname]
```

**INTERNAL
COMMANDS**

The following commands are internal interfaces, for use only by Ixia. Use of these commands may produce undesirable results and are not guaranteed to be backward compatible in future releases:

exists, getHelp, getType, getValidRange, getValidValues, getValidateProc

C

High-Level API

This chapter provides the arguments to set high-level APIs and the list of high-level APIs that are used in IxOS setup.

Arguments to the high-level APIs are passed in one of the following ways:

- **By value:** Denoted by (By value) in the description. By value arguments are either a constant or a \$variable reference. For example:
 `{{1 1 1} {1 2 1}} -or- $portList`
- **By reference:** Denoted by (By reference) in the description. By reference arguments must be references to variables, **without** the '\$'. For example, `pl` after `set pl {{1 1 1} [1 1 2]}`.

Almost all commands return a value of 0 on successful operation. This can be symbolically referred to as `$TCL_OK` in a global context or `$TCL_OK` otherwise. In the examples in this section, a value of 0 is used.

Similarly predefined quantities such as `one2oneArray` are defined in the global context. If your program is running in other than the global context then it is necessary to include a double colon () before the constant or variable name. For example, `one2oneArray`.

NAME - `getAllPorts`

getAllPorts - Gets a list of all ports associated with a port map

SYNOPSIS `getAllPorts portList`

DESCRIPTION The `getAllPorts` returns a list of all ports associated with a port map.

ARGUMENTS**mapName** (By reference) One of the following:

one2oneArray, one2manyArray, many2oneArray, many2manyArray

RETURNS**list** A list of all transmit and receive ports associated with the map. The format of the returned value is a list of lists, for example, {1 1 1} {1 1 2} {1 1 3} {1 1 4}.

EXAMPLES

SEE ALSO *getRxPorts, getTxPorts, map*

NAME - `getRxPorts`

`getRxPorts` - Gets all receive ports associated with a port map

SYNOPSIS

`getRxPorts portList`

DESCRIPTION

The `getRxPorts` returns a list of all receive ports associated with a port map.

ARGUMENTS

mapName

(By reference) One of the following:

one2oneArray, one2manyArray, many2oneArray, many2manyArray

RETURNS

list

A list of all receive ports associated with the map. The format of the returned value is a list of lists, for example, {1 1 1} {1 1 2} {1 1 3} {1 1 4}.

EXAMPLES

SEE ALSO

getAllPorts, getTxPorts, map

NAME - *getTxPorts*

getTxPorts - Gets all transmit ports associated with a port map

SYNOPSIS *getTxPorts portList*

DESCRIPTION The *getTxPorts* returns a list of all transmit ports associated with a port map.

ARGUMENTS

mapName (By reference) One of the following:

one2oneArray, one2manyArray, many2oneArray, many2manyArray

RETURNS

list A list of all transmit ports associated with the map. The format of the returned value is a list of lists, for example, {1 1 1} {1 1 2} {1 1 3} {1 1 4}.

EXAMPLES

SEE ALSO *getAllPorts, getRxPorts, map*

NAME - issuePcpuCommand

issuePcpuCommand - Execute a command on a list of ports

SYNOPSIS

issuePcpuCommand *portList command*

DESCRIPTION

The *issuePcpuCommand* command executes a Linux commands on a set of ports. The result of the command's execution indicates whether the command was sent to the ports or not. No indication is given that the ports actually ran successfully on the ports. The individual port by port result of the command can be retrieved by using the *getFirst* / *getNext* functions of *pcpuCommandService*.

ARGUMENTS

command

The text of the command to be executed, which must use an absolute path. For example, '/bin/ls'. No filename expansion is performed on the command; that is, '/bin/ls /bin/ix*' finds no matches. This, and the restriction on absolute path, may be avoided by executing the command through a *bash* shell, as in:

```
issuePcpuCommand portList "/bin/bash -c 'ls -l /bin/ix*'"
```

portList

(By reference) The list of ports to execute *command* on, in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray

Or a reference to a list. For example, *pl* after

```
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}
```

RETURNS

0

The commands were sent to the ports.

1

The commands could not be sent to the ports.

EXAMPLES

```
package require IxTclServices

set host    techpubs-400

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```
set portList [list [list 1 1 1] [list 1 1 2]]  
  
set ret [issuePcpuCommand portList "/bin/bash -c 'rm /tmp/hello'"]  
ixPuts "Return is $ret"  
  
for {set next [pcpuCommandService getFirst]} \  
{$next != $::TCL_ERROR} \  
{set next [pcpuCommandService getNext]} {  
set chassis [pcpuCommandService cget -chassisID]  
set card [pcpuCommandService cget -cardID]  
set port [pcpuCommandService cget -portID]  
set command [pcpuCommandService cget -command]  
set output [pcpuCommandService cget -output]  
set result [pcpuCommandService cget -result]  
  
ixPuts "$chassis:$card:$port, cmd: $command, result: $result,  
output: $output"  
}
```

SEE ALSO

pcpuCommandService

NAME - ixAbortPoeArm

ixAbortPoeArm - abort the arming of a list of PoE ports

SYNOPSIS *ixAbortPoeArm portList*

DESCRIPTION The **ixAbortPoeArm** command abort the arming of a list of PoE ports.

ARGUMENTS

portList (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after

set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0 Successful.

1 An error occurred.

EXAMPLES

SEE ALSO *ixAbortPortPoeArm*, *ixArmPoeTrigger*, *ixArmPortPoeTrigger*

NAME - ixAbortPortPoeArm

ixAbortPortPoeArm - abort the arming of an individual PoE port

SYNOPSIS *ixAbortPortPoeArm chassisID cardID portID*

DESCRIPTION The **ixAbortPortPoeArm** command aborts the arming of an individual PoE port.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

RETURNS

0 No error.

1 Error.

EXAMPLES

SEE ALSO *ixAbortPoeArm, ixArmPoeTrigger, ixArmPortPoeTrigger*

NAME - ixArmPoeTrigger

ixArmPoeTrigger - arm a list of PoE ports for triggering

SYNOPSIS *ixArmPoeTrigger portList*

DESCRIPTION The **ixArmPoeTrigger** command arms a list of PoE ports for triggering.

ARGUMENTS

portList (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after

set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0 Successful.

1 An error occurred.

EXAMPLES

SEE ALSO *ixAbortPoeArm*, *ixAbortPortPoeArm*, *ixArmPortPoeTrigger*

NAME - ixArmPortPoeTrigger

ixArmPortPoeTrigger - arm an individual PoE port for trigger

SYNOPSIS *ixArmPortPoeTrigger chassisID cardID portID*

DESCRIPTION The **ixArmPortPoeTrigger** command arms an individual PoE port.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

RETURNS

0 No error.

1 Error.

EXAMPLES

SEE ALSO *ixAbortPoeArm, ixAbortPortPoeArm, ixArmPoeTrigger*

NAME - ixCheckLinkState

ixCheckLinkState - checks the link state on a group of ports

SYNOPSIS

ixCheckLinkState *portList*

DESCRIPTION

The **ixCheckLinkState** command checks the link state on a group of ports. This command must be called in the beginning of the script to ensure that all links are up before any traffic is transmitted to the DUT.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

Links on all ports are up.

1

Link on one or more ports is down.

EXAMPLES

```
package req IxTclHal

set host      galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chassis [ixGetChassisID $host]

set cardA    1
set portA   1
set cardB    1
set portB   2
set portList [list [list $chassis $cardA $portA] [list $chassis
$cardB $portB]]
```

```

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chassis $cardA $portA   $chassis $cardB $portB
map add      $chassis $cardB $portB   $chassis $cardA $portA

port setDefault
port set $chassis $cardA $portA
port set $chassis $cardB $portB

stream setDefault
stream config -dma      stopStream
stream config -numFrames 100000
stream set     $chassis $cardA $portA 1

stream config -numFrames 200000
stream set     $chassis $cardB $portB 1

ixWritePortsToHardware one2oneArray

# wait for write ports to have an effect
after 1000

if {[ixCheckLinkState one2oneArray] != 0} {
    ixPuts "One or more links are down"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

NAME - ixCheckOwnership

ixCheckOwnership - checks the ownership for a list of ports

SYNOPSIS

ixCheckOwnership *portList*

DESCRIPTION

The **ixCheckOwnership** command checks the ownership on a list of ports; the port list must be passed by value. It accepts * as a wild card to indicate all cards or all ports on a card. A wild card cannot be used for chassis ID. Also, if a combination of a list element containing wild cards and port numbers are passed, then the port list passed MUST be in a sorted order, otherwise the some of those ports might not make it in the list.

ARGUMENTS

portList

(By value) The list of ports in one of the following formats:

One of the following literal strings, or a reference to a variable with the \$ (for example, \$pl after set pl ...)
 {{1 1 1}}
 {{1 1 1} {1 1 2} {1 1 3} {1 1 4}}
 {{1 1 *} {1 2 1} {1 2 2}}
 {1,1,* 1,2,1 1,2,2}

RETURNS

0

All of the ports are available for the ‘taking’.

100

One or more of the ports are owned by someone else.

EXAMPLES

```
package req IxTclHal

set host      galaxy
set username George
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set portListG [list [list $chas 2 2]]
```

```
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portListG force] {
    ixPuts $::ixErrorInfo
    return 1
}

# Login Bill and make a port list for all ports on cards 1 and 2
ixLogin bill
set portListB [list [list $chassis 1 *] [list $chassis 2 *]]

# This should fail because 1, 2, 2 is owned by George
if {[ixCheckOwnership $portListB] == 0} {
    ixPuts "Ports $portListB are available"
} else {
    ixPuts "One or more of $portListB are unavailable"
}

# Now we'll avoid that port and express the list a different way
set portListB [list 1,1,* 1,2,1]

if {[ixCheckOwnership $portListB] == 0} {
    ixPuts "Ports $portListB are available"
} else {
    ixPuts "One or more of $portListB are unavailable"
}

# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

*ixClearOwnership, ixLogin, ixLogout, ixPortClearOwnership,
ixPortTakeOwnership, ixTakeOwnership*

NAME - ixCheckPPPState

ixCheckPPPState - checks the PPP state on a group of POS ports

SYNOPSIS

ixCheckPPPState *portList [message]*

DESCRIPTION

The **ixCheckPPPState** command checks the PPP state of all PoS ports in a group of ports in parallel and labels the ones that are down. Then it polls the links that are down for two seconds and returns 1 if any port is still down and a 0 if all ports are up.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

message

(By value) (*Optional, default = messageOn*) Indicates that a message with the ports' state is to be written to STDOUT or not.

RETURNS

0

Links on all ports are up.

1

Link on one or more ports is down.

EXAMPLES

```
package req IxTclHal

set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chassis [ixGetChassisID $host]

set cardA    2
set portA   1
```

```
set cardB    2
set portB   2
set portList [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chassis $cardA $portA   $chassis $cardB $portB
map add      $chassis $cardB $portB   $chassis $cardA $portA

if {[ixCheckPPPState one2oneArray] != 0} {
    ixPuts "PPP is down"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

NAME - SixCheckPortTransmitDone

ixCheckPortTransmitDone - checks whether transmission is done on a port

SYNOPSIS

ixCheckPortTransmitDone *chassisID cardID portID*

DESCRIPTION

The **ixCheckPortTransmitDone** command polls the transmit rate statistic counter and returns when transmission has stopped. **Note:** This command should be called no earlier than one second after starting transmit with *ixStartTransmit* or *ixStartPortTransmit*.

Note: It should be preceded by an *after 1000* statement following the previous command, to allow the effects of the previous command to have an effect on the port hardware.

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.

RETURNS

0	No frames were sent or the <i>stat get framesSent</i> command failed.
numTxFrames	No Error; number of frames transmitted since the last time statistics were cleared.

EXAMPLES

```

package require IxTclHal
set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set chas    1
set cardA   1

```

```

set portA    4
set portList [list [list $chassis $cardA $portA]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

port setDefault
port set $chassis $cardA $portA

stream setDefault
stream config -dma          stopStream
stream config -numFrames 100000
stream set      $chassis $cardA $portA 1

ixWritePortsToHardware portList
after 1000
if {[ixCheckLinkState portList] != 0} {
    ixPuts "Link is not up"
}

# Start transmit and wait a bit
ixStartPortTransmit $chassis $cardA $portA
after 1000

# Check if the port has stopped
ixCheckPortTransmitDone $chassis $cardA $portA
ixPuts "PortA Stopped transmitting"

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixCheckTransmitDone*

NAME - ixCheckTransmitDone

ixCheckTransmitDone - checks whether transmission is done on a group of ports

SYNOPSIS

ixCheckTransmitDone *portList*

DESCRIPTION

The **ixCheckTransmitDone** command polls the transmit rate statistic counter and returns when transmission is stopped. **Note:** This command should be called no earlier than one second after starting transmit with *ixStartTransmit* or *ixStartPortTransmit*.

Note: It should be preceded by an *after 1000* statement following the previous command, to allow the effects of the previous command to have an effect on the port hardware.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0 Success

1 Failure.

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
```

```

set cardA 1
set portA 1
set cardB 1
set portB 2

# Examples of four ways to make a port list
set portList1 [list $chas,$cardA,$portA]
set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList2] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type oneZone
map config -type oneZone
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

port setDefault
port set $chas $cardA $portA
port set $chas $cardB $portB

stream setDefault
stream config -dma          stopStream
stream config -numFrames 100000
stream set      $chas $cardA $portA 1

stream config -numFrames 200000
stream set      $chas $cardB $portB 1

ixWritePortsToHardware oneZoneArray
after 1000
if {[ixCheckLinkState oneZoneArray] != 0} {
    ixPuts "Link is not up"
}

# Start transmit and wait a bit
ixStartTransmit oneZoneArray
after 1000

# Check if the first port has stopped
ixCheckTransmitDone portList1
ixPuts "PortA Stopped transmitting"

# Check if both ports have stopped
ixCheckTransmitDone portList2
ixPuts "PortA & PortB Stopped transmitting"

ixStartTransmit oneZoneArray
after 1000

```

```
# Check if both ports have stopped, a different way
ixCheckTransmitDone portList3
ixPuts "PortA & PortB Stopped transmitting"

ixStartTransmit one2oneArray
after 1000

# Check if both ports have stopped, yet another way
ixCheckTransmitDone portList4
ixPuts "PortA & PortB Stopped transmitting"

ixStartTransmit one2oneArray
after 1000
ixCheckTransmitDone one2oneArray
ixPuts "PortA & PortB Stopped transmitting"

# Let go of the ports that we reserved
ixClearOwnership $portList2
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

SixCheckPortTransmitDone

NAME - ixClearArpTable

ixClearArpTable - clears the ARP table on a group of ports simultaneously

SYNOPSIS ixClearArpTable *portList*

DESCRIPTION The **ixClearArpTable** command clears the ARP table by the protocol server.

ARGUMENTS

ixClearArpTable (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after

set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package req IxTclHal

set host      galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA  2
set portA  1
set cardB  2
set portB  2
set portList [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
# Login before taking ownership
if [ixLogin $username] {
```

```
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type oneZone
map config -type oneZone
map add      $chassis $cardA $portA  $chassis $cardB $portB
map add      $chassis $cardB $portB  $chassis $cardA $portA

if {[ixClearArpTable oneZoneArray] != 0} {
    ixPuts "ARP table could not be cleared"
} else {
    ixPuts "ARP table cleared"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixClearPortArpTable

NAME - ixClearOwnership

ixClearOwnership - clears ownership of all the ports in the list

SYNOPSIS ixClearOwnership *[portList] [takeType]*

DESCRIPTION The **ixClearOwnership** command clears ownership of all the ports in the list.

ARGUMENTS

portList (By value) The list of ports in one of the following formats:

One of the following literal strings, or a reference to a variable with the \$
 (for example, \$pl after *set pl ...*)
 {{1 1 1}}
 {{1 1 1} {1 1 2} {1 1 3} {1 1 4}}
 {{1 1 *} {1 2 1} {1 2 2}}
 {1,1,* 1,2,1 1,2,2}

A value of ““(default) clears ownership of all Tcl owned ports.

takeType (By value) (*Optional*) Valid values:

force: take regardless of whether the port is owned by someone else
notForce: (default) do not force ownership

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package req IxTclHal

set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
```

```
set portList1 {{1 1 1}}
set portList2 {{1 1 1} {1 1 2} {1 1 3} {1 1 4}}
set portList3 {{1 1 *} {1 2 1} {1 2 2} }

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}

if {[ixClearOwnership $portList1] != 0} {
    ixPuts "Could not clear ownership for $portList1\n"
}
if {[ixClearOwnership $portList2] != 0} {
    ixPuts "Could not clear ownership for $portList2\n"
}
if {[ixClearOwnership $portList3 notForce] != 0} {
    ixPuts "Could not clear ownership for $portList3\n"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixTakeOwnership, *ixPortClearOwnership*, *ixPortTakeOwnership*

NAME - ixClearPacketGroups

ixClearPacketGroups - clears the packet group statistics of all the ports in the list

SYNOPSIS	ixClearPacketGroups [<i>portList</i>]
-----------------	---

DESCRIPTION	The ixClearPacketGroups command clears the packet group statistics of all the ports in the list.
--------------------	---

ARGUMENTS

portList	(By value) The list of ports in one of the following formats:
-----------------	---

One of the following literal strings, or a reference to a variable with the \$ (for example, \$pl after set pl ...)

```

{{1 1 1}}
{{1 1 1} {1 1 2} {1 1 3} {1 1 4}}
{{1 1 *} {1 2 1} {1 2 2}}
{1,1,* 1,2,1 1,2,2}

```

A value of ““(default) clears ownership of all Tcl owned ports.

RETURNS

0	No error; the command was successfully delivered to the IxServer.
----------	---

1	Error; the command was delivered to the IxServer but it could not process the message.
----------	--

EXAMPLES

```

package require IxTclHal

set host      galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set chas    1
set cardA   1

```

```

set portA    1
set cardB   1
set portB   2

# Four different port list formats
set portList1 [list $chas,$cardA,$portA]
set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

# Try each of the formats
if {[ixClearPacketGroups portList1] != 0} {
    ixPuts $::ixErrorInfo
    return 1
}
if {[ixClearPacketGroups portList2] != 0} {
    ixPuts $::ixErrorInfo
    return 1
}
if {[ixClearPacketGroups portList3] != 0} {
    ixPuts $::ixErrorInfo
    return 1
}
if {[ixClearPacketGroups portList4] != 0} {
    ixPuts $::ixErrorInfo
    return 1
}
if {[ixClearPacketGroups one2oneArray] != 0} {
    ixPuts $::ixErrorInfo
    return 1
}

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
return 0

```

SEE ALSO*ixClearPortPacketGroups*

NAME - ixClearPerStreamTxStats

ixClearPerStreamTxStats - Clear per stream Tx statistics counters on the *portList*.

SYNOPSIS

ixClearPerStreamTxStats *portList*

DESCRIPTION

The **ixClearPerStreamTxStats** command clears the per stream statistics for the specified port.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

Successful.

1

An error occurred.

EXAMPLES

SEE ALSO

ixClearStats, *ixClearPortStats*

NAME - **ixClearPortArpTable**

ixClearPortArpTable - clears the ARP table on an individual port

SYNOPSIS **ixClearPortArpTable** *chassisID cardID portID*

DESCRIPTION The **ixClearPortArpTable** command clears the ARP table on a single port.

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.

RETURNS

0	No error; the command was successfully delivered to the IxServer.
1	Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```

package require IxTclHal
set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card   1
set port   1
set portList [list [list $chas $card $port]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use

```

```
if [ixTakeOwnership $portList] {  
    ixPuts $::ixErrorInfo  
    return 1  
}  
  
if {[ixClearPortArpTable $chassis $card $port] != 0} {  
    ixPuts "Could not clear Arp table on $chassis:$card:$port"  
}  
  
# Let go of the ports that we reserved  
ixClearOwnership $portList  
# Disconnect from the chassis we're using  
ixDisconnectFromChassis $host  
# If we're running on UNIX, disconnect from the TCL Server  
if [isUNIX] {  
    ixDisconnectTclServer $host  
}
```

SEE ALSO

ixClearArpTable

NAME - ixClearPortPacketGroups

ixClearPortPacketGroups - zero all packet group counters on an individual port

SYNOPSIS	ixClearPortPacketGroups <i>chassisID cardID portID</i>
-----------------	--

DESCRIPTION	The ixClearPortPacketGroups command clears all packet group counters on a single port.
--------------------	---

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.

RETURNS

0	No error; the command was successfully delivered to the IxServer.
1	Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA 1
set portA 1
set portList [list [list $chas $cardA $portA]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
```

```
        return 1
    }
    # Take ownership of the ports we'll use
    if [ixTakeOwnership $portList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    if {[ixClearPortPacketGroups $chas $cardA $portA] != 0} {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Let go of the ports that we reserved
    ixClearOwnership $portList
    # Disconnect from the chassis we're using
    ixDisconnectFromChassis $host
    # If we're running on UNIX, disconnect from the TCL Server
    if [isUNIX] {
        ixDisconnectTclServer $host
    }
    return 0
```

SEE ALSO

ixClearPacketGroups

NAME - ixClearPortStats

ixClearPortStats - zero all statistic counters on an individual port

SYNOPSIS ixClearPortStats *chassisID cardID portID*

DESCRIPTION The **ixClearPortStats** command clears all statistic counters on a single port.

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.

RETURNS

0	No error; the command was successfully delivered to the IxServer.
1	Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA  1
set portA 1
set portList [list [list $chas $cardA $portA]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
```

```
if [ixTakeOwnership $portList] {  
    ixPuts $::ixErrorInfo  
    return 1  
}  
  
if {[ixClearPortStats $chas $cardA $portA] != 0} {  
    ixPuts "Could not clear time stamp for $chas:$cardA:$portA"  
}  
  
# Let go of the ports that we reserved  
ixClearOwnership $portList  
# Disconnect from the chassis we're using  
ixDisconnectFromChassis $host  
# If we're running on UNIX, disconnect from the TCL Server  
if [isUNIX] {  
    ixDisconnectTclServer $host  
}
```

SEE ALSO*ixClearStats*

NAME - *ixClearScheduledTransmitTime*

ixClearScheduledTransmitTime - clears the amount of transmit time for a port list

SYNOPSIS *ixClearScheduledTransmitTime portList*

DESCRIPTION Clears the maximum amount of time that a group of ports transmits. This is only valid for ports that support the *portFeatureScheduledTxDuration* feature, which may be tested with the *port isvalidFeature* command.

ARGUMENTS

portList (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
set portList {{1 1 1} {1 1 2}}
if [ixClearScheduledTransmitTime portList] {
    ixPuts $::ixErrorInfo
}
```

SEE ALSO *::ixSetScheduledTransmitTime*

NAME - ixClearStats

ixClearStats - zero all statistic counters on a group of ports simultaneously

SYNOPSIS

ixClearStats *portList*

DESCRIPTION

The **ixClearStats** command clears all statistic counters on a list of ports simultaneously. This command must be called before the transmission of validation traffic is started so that the proper metrics can be calculated at the end of transmission.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set chas    1
set cardA   1
set portaA  1
```

```

set cardB    1
set portB   2

# Four different port list formats
set portList1 [list $chassis,$cardA,$portA]
set portList2 [list $chassis,$cardA,$portA $chassis,$cardB,$portB]
set portList3 [list [list $chassis $cardA $portA] [list $chassis $cardB
$portB]]
set portList4 [list [list $chassis,$cardA,$portA] [list
$chassis,$cardB,$portB] ]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chassis $cardA $portA   $chassis $cardB $portB
map add      $chassis $cardB $portB   $chassis $cardA $portA

# Try each of the formats
if {[ixClearStats portList1] != 0} {
    ixPuts "Could not clear time stamp for $portList1"
}
if {[ixClearStats portList2] != 0} {
    ixPuts "Could not clear time stamp for $portList2"
}
if {[ixClearStats portList3] != 0} {
    ixPuts "Could not clear time stamp for $portList3"
}
if {[ixClearStats portList4] != 0} {
    ixPuts "Could not clear time stamp for $portList4"
}
if {[ixClearStats one2oneArray] != 0} {
    ixPuts "Could not clear time stamp for $one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixClearPortStats*

NAME - ixClearTimeStamp

ixClearTimeStamp - synchronizes the timestamp value among all chassis

SYNOPSIS

ixClearTimeStamp *portList*

DESCRIPTION

The **ixClearTimeStamp** command sends a message to the IxServer to synchronize the timestamp on a group of chassis. This feature is useful for calculating latency on ports across chassis.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set chas    1
set cardA  1
set portA  1
set cardB  1
```

```
set portB    2

# Four different port list formats
set portList1 [list $chassis,$cardA,$portA]
set portList2 [list $chassis,$cardA,$portA $chassis,$cardB,$portB]
set portList3 [list [list $chassis $cardA $portA] [list $chassis $cardB $portB]]
set portList4 [list [list $chassis,$cardA,$portA] [list
$chassis,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chassis $cardA $portA      $chassis $cardB $portB
map add      $chassis $cardB $portB      $chassis $cardA $portA

# Try each of the formats
if {[ixClearTimeStamp portList1] != 0} {
    ixPuts "Could not clear time stamp for $portList1"
}
if {[ixClearTimeStamp portList2] != 0} {
    ixPuts "Could not clear time stamp for $portList2"
}
if {[ixClearTimeStamp portList3] != 0} {
    ixPuts "Could not clear time stamp for $portList3"
}
if {[ixClearTimeStamp portList4] != 0} {
    ixPuts "Could not clear time stamp for $portList4"
}
if {[ixClearTimeStamp one2oneArray] != 0} {
    ixPuts "Could not clear time stamp for $one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

NAME - ixCollectStats

ixCollectStats - collect a particular statistic on a group of ports

SYNOPSIS

ixCollectStats *rxList statName rxStats totalStats*

DESCRIPTION

The **ixCollectStats** command gathers the same specified statistic from a number of ports and places the results in a return array.

ARGUMENTS

rxList

(By value) The list of ports in one of the following formats:

One of the following literal strings, or a reference to a variable with the \$ (for example, \$pl after set pl ...)
 {{1 1 1}}
 {{1 1 1} {1 1 2} {1 1 3} {1 1 4}}
 {{1 1 *} {1 2 1} {1 2 2}}
 {1,1,* 1,2,1 1,2,2}

statName

(By value or reference) The name of the statistic to poll. This has to match one of the standard options defined in the *stat* command.

rxStats

(By reference) The array containing the returned statistics per port. Each element is accessed with three comma separated arguments corresponding to the chassis, card and port being accessed. For example, \$rxStats(1, 1, 1)

totalStats

(By reference) The total of the values in *RxStats*.

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
```

```

        ixPuts $::ixErrorInfo
        return 1
    }

    # Get the chassis ID to use in port lists
    set chas [ixGetChassisID $host]

    set chas    1
    set cardA   1
    set portA   1
    set cardB   1
    set portB   2

    set portList [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
    # Login before taking ownership
    if [ixLogin $username] {
        ixPuts $::ixErrorInfo
        return 1
    }
    # Take ownership of the ports we'll use
    if [ixTakeOwnership $portList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Setup start
    map new -type one2one
    map config -type one2one
    map add      $chas $cardA $portA  $chas $cardB $portB
    map add      $chas $cardB $portB  $chas $cardA $portA

    port setDefault
    port set $chas $cardA $portA
    port set $chas $cardB $portB

    stream setDefault
    stream config -dma          stopStream
    stream config -numFrames 100000
    stream set      $chas $cardA $portA 1

    stream config -numFrames 200000
    stream set      $chas $cardB $portB 1

    # Set up the ports
    ixWritePortsToHardware one2oneArray
    after 1000
    if {[ixCheckLinkState one2oneArray] != 0} {
        ixPuts "Link is not up"
        exit
    }

    # Clear statistics before starting
    if {[ixClearStats portList] != 0} {
        ixPuts "Could not clear statistics on $portList"
    }

    # Start transmit and wait a bit
    ixStartTransmit one2oneArray
    after 1000

    # Check if the both ports have stopped
    ixCheckTransmitDone portList

```

```
ixPuts "Ports stopped transmitting"

if {[ixCollectStats $portList framesSent myArray myTotal] != 0} {
    ixPuts "Could not collect statistics on $portList"
}

ixPuts "Total number is $myTotal"

foreach p $portList {
    scan $p "%d %d %d" ch ca po
    ixPuts "Port $p is $myArray($ch,$ca,$po)"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

NAME - ixConnectToChassis

ixConnectToChassis - connects to a list of chassis

SYNOPSIS ixConnectToChassis *chassisList [cableLength]*

DESCRIPTION The **ixConnectToChassis** command connects to a list of chassis given the hostnames or IP addresses.

ARGUMENTS

chassisList (By value) The list of chassis hostnames or IP addresses, called by value.

cableLength (By value) (Optional) The length of the sync cable that connects the chain of chassis. Valid values are:

Option	Value	Usage
<i>cable3feet</i>	0	default
<i>cable6feet</i>	1	
<i>cable9feet</i>	2	
<i>cable12feet</i>	3	
<i>cable15feet</i>	4	
<i>cable18feet</i>	5	
<i>cable21feet</i>	6	
<i>cable24feet</i>	7	

RETURNS

- 0** No Error, connection was established with the IxServer.
- 1** Error connecting to IxServer; possible causes are invalid hostname or IP address for chassis, IxServer not running on the chassis, or other network problem.
- 2** Version mismatch.
- 3** Timeout connecting to chassis; possible causes are invalid hostname or IP address for chassis, or IxServer not running on the chassis.

EXAMPLES

```
package require IxTclHal

set host1    localhost
set host2    galaxy

set ret [ixConnectToChassis $host1]
switch $ret {
    1 {ixPuts "Error connecting to chassis"}
    2 {ixPuts "Version mismatch with chassis"}
    3 {ixPuts "Timeout connecting to chassis"}
}
```

```
ixDisconnectFromChassis

set pl [list $host1 $host2]
set ret [ixConnectToChassis $pl 1]
switch $ret {
    1 {ixPuts "Error connecting to chassis"}
    2 {ixPuts "Version mismatch with chassis"}
    3 {ixPuts "Timeout connecting to chassis"}
}
ixDisconnectFromChassis
```

SEE ALSO

ixDisconnectFromChassis, ixConnectToChassis, ixProxyConnect

NAME - *ixConvertFromSeconds*

ixConvertFromSeconds - convert a number of seconds to hours, minutes and seconds

SYNOPSIS *ixConvertFromSeconds time hours minutes seconds*

DESCRIPTION This command converts a number of seconds into hours, minutes and seconds.

ARGUMENTS

time The time, in seconds, to be converted.

hours This argument is accessed by reference; that is, the name of a TCL variable. This is the number of hours in *time*.

minutes This argument is accessed by reference; that is, the name of a TCL variable. This is the number of minutes in *time*.

seconds This argument is accessed by reference; that is, the name of a TCL variable. This is the number of seconds in *time*.

RETURNS

none

EXAMPLE

```
set time 10000
set hours 0
set minutes 0
set seconds 0
ixConvertFromSeconds $time hours minutes seconds
```

SEE ALSO *ixConvertToSeconds*

NAME - ixConnectToTclServer

ixConnectToTclServer - connect a Unix client to the IxTclServer

SYNOPSIS

ixConnectToTclServer *serverName*

DESCRIPTION

The **ixConnectToTclServer** command connects a Tcl Client running on a non-Windows workstation to the IxTclServer running on a chassis or Windows-based system.

ARGUMENTS

serverName

(By value) The name or IP address of the machine running the IxTclServer, called by value.

RETURNS

0

No Error, connection was established with the IxTclServer.

1

Error of any type.

EXAMPLES

```
package require IxTclHal

set host    galaxy

if {[ixConnectToTclServer $host] != 0} {
    ixPuts "Could not connect to Tcl Server on $host"
}

ixDisconnectTclServer
```

SEE ALSO

ixConnectToChassis, ixProxyConnect

NAME - ixConvertToSeconds

ixConvertToSeconds - convert hours, minutes and seconds to a number of seconds

SYNOPSIS *ixConvertToSeconds hours minutes seconds*

DESCRIPTION This command converts a number hours, minutes and seconds into seconds.

ARGUMENTS

hours The hours to be converted.

minutes The minutes to be converted.

seconds The hours to be converted.

RETURNS

time The time, in seconds.

EXAMPLE *ixConvertToSeconds 2 46 40*

Returns 10000

SEE ALSO *ixConvertFromSeconds*

NAME - ixCreatePortListWildCard

ixCreatePortListWildCard - creates a port list using wildcard '*' specification for cards and/or ports

SYNOPSIS

ixCreatePortListWildCard *portList [excludeList]*

DESCRIPTION

The **ixCreatePortListWildCard** command creates a list of ports in a sorted order based on the physical slots. Both arguments are passed by value. It accepts '*' as a wild card to indicate all cards or all ports on a card. A wild card cannot be used for chassis ID. Also, if a combination of a list element containing wild cards and port numbers is passed, then the port list passed MUST be in a sorted order, otherwise the some of those ports might not make it in the list.

ARGUMENTS

portList

(By value) The list of ports in one of the following formats:

One of the following literal strings, or a reference to a variable with the \$
 (for example, \$pl after set pl ...)
 {{1 1 1}}
 {{1 1 1} {1 1 2} {1 1 3} {1 1 4}}
 {{1 1 *} {1 2 1} {1 2 2}}
 {1,1,* 1,2,1 1,2,2}

excludeList

(By value) The list of ports to exclude in one of the following formats. No wildcard may be used in this list:

One of the following literal strings, or a reference to a variable with the \$
 (for example, \$pl after set pl ...)
 {{1 1 1}}
 {{1 1 2} {1 1 3} {1 1 4}}

RETURNS

A list of lists with the expanded port list.

EXAMPLES

```

package require IxTclHal
set host      galaxy

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

```

```
set portList    { {1 1 *} {1 * 2} }
set excludeList { {1 1 1} {1 1 2} {1 2 2} }

set retList [ixCreatePortListWildCard $portList]
ixPuts $retList
set retList [ixCreatePortListWildCard $portList $excludeList]
ixPuts $retList
```

SEE ALSO*ixCreateSortedPortList*

NAME - ixCreateSortedPortList

ixCreateSortedPortList - creates a port list for a range of ports, excluding specified ports

SYNOPSIS

ixCreateSortedPortList *portFrom* *portTo* *excludeList*

DESCRIPTION

The **ixCreateSortedPortList** command creates a sorted list of ports based on the range of ports passed.

ARGUMENTS

portFrom

(By value) The first port number. For example, {1 1 1}.

portTo

(By value) The last port number. For example, {1 5 4}.

excludeList

(By value) A list of lists containing individual ports to be excluded from the list. For example, {{1 3 1} {1 3 2}}

EXAMPLES

```
package require IxTclHal

set host galaxy
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

set from {1 1 1}
set to {1 2 1}
set ex {{1 1 4} }

set retList [ixCreateSortedPortList $from $to $ex]
ixPuts $retList
```

RETURNS

A sorted list of lists with the expanded port list.

SEE ALSO

ixCreatePortListWildCard

NAME - **ixDisableArpResponse**

ixDisableArpResponse - Disable ARP response on a set of ports

SYNOPSIS	ixDisableArpResponse <i>portList</i>
-----------------	--------------------------------------

DESCRIPTION	The ixDisableArpResponse disables the ARP response engine for the set of ports.
--------------------	--

ARGUMENTS

portList	(By reference) The list of ports in one of the following formats:
-----------------	---

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0	No error; the command was successfully delivered to the IxServer.
----------	---

1	Error; the command was delivered to the IxServer but it could not process the message.
----------	--

EXAMPLES

```
package require IxTclHal

set host      galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set chas    1
set cardA  1
set portA  1
set cardB  1
set portB  2
```

```

# Four different port list formats
set portList1 [list $chas,$cardA,$portA]
set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type oneZone
map config -type oneZone
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

# Try each of the formats
if {[ixDisableArpResponse portList1] != 0} {
    ixPuts "Could not disable ARP response for $portList1"
}
if {[ixDisableArpResponse portList2] != 0} {
    ixPuts "Could not disable ARP response for $portList2"
}
if {[ixDisableArpResponse portList3] != 0} {
    ixPuts "Could not disable ARP response for $portList3"
}
if {[ixDisableArpResponse portList4] != 0} {
    ixPuts "Could not disable ARP response for $portList4"
}
if {[ixDisableArpResponse oneZoneArray] != 0} {
    ixPuts "Could not disable ARP response for $oneZoneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

ixDisablePortArpResponse, *ixEnableArpResponse*, *ixEnablePortArpResponse*

NAME - ixDisablePortArpResponse

ixDisablePortArpResponse - Disable ARP response on a single port

SYNOPSIS

ixDisableArpResponse *chassisID cardID portID [write]*

DESCRIPTION

The **ixDisablePortArpResponse** disables the ARP response engine for the port.

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.
write	(By value) (<i>Optional</i>) Valid values: <i>write</i> : the action is committed to hardware <i>noWrite</i> : the action is not committed to hardware but just set in IxHAL <i>(default)</i>

RETURNS

0	No error; the command was successfully delivered to the IxServer.
1	Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA 1
```

```
set portA    1
set portList [list [list $chas $cardA $portA]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
ip setDefault
ip set $chas $cardA $portA

if {[ixEnablePortArpResponse $::oneIpToOneMAC $chas $cardA $portA]
!= 0} {
    ixPuts "Could not enable ARP response for $chas:$cardA:$portA"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixDisableArpResponse, *ixEnableArpResponse*, *ixEnablePortArpResponse*

NAME - *ixDisconnectFromChassis*

ixDisconnectFromChassis - disconnects from all chassis connected

SYNOPSIS	<i>ixDisconnectFromChassis [chassis ...]</i>
-----------------	--

DESCRIPTION	The ixDisconnectFromChassis command is called at the end of the script which disconnects from all the chassis that were connected to in the beginning of the script. It also frees any memory allocated by the Tcl script by calling the cleanUp command.
--------------------	--

ARGUMENTS

chassis	(By value) (<i>Optional</i>) A variable number of chassis to disconnect from.
----------------	---

RETURNS

0	No Error, successfully disconnected.
----------	--------------------------------------

EXAMPLES	<pre>package require IxTclHal set host galaxy set username user # Check if we're running on UNIX - connect to the TCL Server # which must be running on the chassis if [isUNIX] { if [ixConnectToTclServer \$host] { ixPuts "Could not connect to \$host" return 1 } } # Now connect to the chassis if [ixConnectToChassis \$host] { ixPuts \$::ixErrorInfo return 1 } # Get the chassis ID to use in port lists set chas [ixGetChassisID \$host] # Disconnect from the chassis we're using if [ixDisconnectFromChassis \$host] { ixPuts \$::ixErrorInfo } # If we're running on UNIX, disconnect from the TCL Server if [isUNIX] { ixDisconnectTclServer \$host }</pre>
-----------------	---

SEE ALSO	<i>ixConnectToChassis, ixConnectToChassis</i>
-----------------	---

NAME - ixDisconnectTclServer

ixDisconnectTclServer - Disconnect a Unix client from the IxTclServer

SYNOPSIS

ixDisconnectTclServer *serverName*

DESCRIPTION

The **ixDisconnectTclServer** command disconnects a Tcl Client running on a non-Windows workstation to the IxTclServer running on a chassis or Windows-based system.

ARGUMENTS

serverName

(By value) This argument is no longer used, but must be present.

RETURNS

0

No Error, successfully disconnected

EXAMPLES

See examples in *ixConnectToTclServer*.

SEE ALSO

ixConnectToTclServer, *ixConnectToChassis*

NAME - ixEnableArpResponse

ixEnableArpResponse - enable ARP response on a set of ports

SYNOPSIS

ixEnableArpResponse *mapType portList*

DESCRIPTION

The **ixEnableArpResponse** gets the MAC and IP address for a set of ports, sets up the address table and enables the ARP response engine for the set of ports. IP configuration must have been performed for this command to succeed.

ARGUMENTS

mapType

(By value) The type of IP to MAC mapping to be used. One of:

Option	Value	Usage
oneIpToOneMAC	0	Each IP address is mapped to a single MAC address.
manyIpToOneMAC	1	All the IP addresses for a port are mapped to a single MAC address.

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,2 1,1,3 1,1,4}

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
```

```

set cardA    1
set portA   1
set cardB    1
set portB   2

# For different port list formats
set portList1 [list $chassis,$cardA,$portA]
set portList2 [list $chassis,$cardA,$portA $chassis,$cardB,$portB]
set portList3 [list [list $chassis $cardA $portA] [list $chassis $cardB
$portB]]
set portList4 [list [list $chassis,$cardA,$portA] [list
$chassis,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chassis $cardA $portA    $chassis $cardB $portB
map add      $chassis $cardB $portB    $chassis $cardA $portA

ip setDefault
ip set 1 1 1
ip set 1 1 2

# Try each of the formats
if {[ixEnableArpResponse oneIpToOneMAC portList1] != 0} {
    ixPuts "Could not enable ARP response for $portList1"
}
if {[ixEnableArpResponse oneIpToOneMAC portList2] != 0} {
    ixPuts "Could not enable ARP response for $portList2"
}
if {[ixEnableArpResponse manyIpToOneMAC portList3] != 0} {
    ixPuts "Could not enable ARP response for $portList3"
}
if {[ixEnableArpResponse manyIpToOneMAC portList4] != 0} {
    ixPuts "Could not enable ARP response for $portList4"
}
if {[ixEnableArpResponse manyIpToOneMAC one2oneArray] != 0} {
    ixPuts "Could not enable ARP response for $one2oneArray"
}
# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

ixDisableArpResponse, *ixDisablePortArpResponse*, *ixEnablePortArpResponse*

NAME - ixEnablePortArpResponse

ixEnablePortArpResponse - enable ARP response on a single port

SYNOPSIS

ixEnableArpResponse *mapType chassisID cardID portID [write]*

DESCRIPTION

The **ixEnablePortArpResponse** gets the MAC and IP address for a single port, sets up the address table and enables the ARP response engine for the port. IP configuration must have been performed for this command to succeed.

ARGUMENTS

mapType

(By value) The type of IP to MAC mapping to be used. One of:

Option	Value	Usage
oneIpToOneMAC	0	Each IP address is mapped to a single MAC address.
manyIpToOneMAC	1	All the IP addresses for a port are mapped to a single MAC address.

chassisID

(By value) The ID number of the chassis.

cardID

(By value) The ID number of the card.

portID

(By value) The ID number of the port.

write

(By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```
}
```

```
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
```

```
set cardA 1
set portA 1
set portList [list [list $chas $cardA $portA]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

ip setDefault
ip set $chas $cardA $portA

if {[ixEnablePortArpResponse $::oneIpToOneMAC $chas $cardA $portA]
!= 0} {
    ixPuts "Could not enable ARP response for $chas:$cardA:$portA"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixDisableArpResponse, *ixDisablePortArpResponse*, *ixEnableArpResponse*

NAME - **ixEnableIntrinsicLatencyAdjustment**

ixEnableIntrinsicLatencyAdjustment - enables the Intrinsic Latency Adjustment on the ports that support the feature

SYNOPSIS

EnableIntrinsicLatencyAdjustment portlist enable write

DESCRIPTION

The **ixEnableIntrinsicLatencyAdjustment** command enables the Intrinsic Latency Adjustment on the ports that support the feature.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

enable

true/false

write

write / nowrite (*default = nowrite*)

RETURNS

-0 OK.

-1 TCL error

-100 Port is not available

-101 Unsupported feature

EXAMPLES

`ixEnableIntrinsicLatencyAdjustment portlist true write`

SEE ALSO

ixEnablePortIntrinsicLatencyAdjustment

ixIsIntrinsicLatencyAdjustmentEnabled

NAME - ixEnablePortIntrinsicLatencyAdjustment

ixEnablePortIntrinsicLatencyAdjustment - enables the Intrinsic Latency Adjustment on the ports that support the feature

SYNOPSIS	<i>ixEnablePortIntrinsicLatencyAdjustment chassisID cardID portID enable write</i>
-----------------	--

DESCRIPTION	The ixEnablePortIntrinsicLatencyAdjustment command enables the Intrinsic Latency Adjustment on the ports that support the feature.
--------------------	---

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.
enable	true/false
write	write / nowrite (<i>default = nowrite</i>)

RETURNS

-0	OK.
-1	TCL error
-100	Port is not available
-101	Unsupported feature

EXAMPLES	ixEnablePortIntrinsicLatencyAdjustment \$chassId \$cardId \$portId true write
-----------------	---

SEE ALSO	<i>ixEnableIntrinsicLatencyAdjustment</i> <i>ixIsIntrinsicLatencyAdjustmentEnabled</i>
-----------------	---

NAME - ixErrorInfo

ixErrorInfo - get the text of the last error

SYNOPSIS

\$::ixErrorInfo

DESCRIPTION

The **ixErrorInfo** global variable holds the text of the last error detected.

EXAMPLES

```
package require IxTclHal

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
```

SEE ALSO

NAME - ixGetChassisID

ixGetChassisID - get the ID of a chassis given its name

SYNOPSIS

ixGetChassisID *chassisName*

DESCRIPTION

The **ixGetChassisID** command gets the ID number assigned to a chassis in the chain.

ARGUMENTS**chassisName**

(By value) The hostname or IP address of chassis.

RETURNS**-1**

The *chassisName* could not be found.

chassisID

The ID number that was assigned to this chassis when a connection to the IxServer was made.

EXAMPLES

```
package require IxTclHal
set host      galaxy
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
set chas [ixGetChassisID $host]
if {$chas < 0} {
    ixPuts "Could not get chassis ID for $host"
} else {
    ixPuts "Chassis ID for $host is $chas"
}
# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixConnectToChassis

NAME - ixGetLineUtilization

ixGetLineUtilization: gets the line utilization in one of two formats

SYNOPSIS

ixGetLineUtilization chasID cardID portID [rateType]

DESCRIPTION

The **ixGetLineUtilization** command returns the line utilization either as a percentage of the maximum value or it terms of frames per second.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

rateType (By value) The requested return format. One of:

Option	Value	Usage
typePercentMaxRate	0	(default) Returns the composite percentage of the maximum rate.
typeFpsRate	1	Returns the frames per second rate.

RETURNS

The value indicated by *rateType*.

EXAMPLES

```
package require IxTclHal  
  
set fps [ixGetLineUtilization 1 1 1 typeFpsRate]
```

SEE ALSO

ixUtils

NAME - **ixGlobalSetDefault**

ixGlobalSetDefault: This command calls for the setDefault for all the IxTclHal commands as a form of initialization.

SYNOPSIS	ixGlobalSetDefault
DESCRIPTION	The ixGlobalSetDefault command sets the default values for all the IxTclHal commands as a form of initialization.
ARGUMENTS	No arguments for this command.
RETURNS	None
EXAMPLES	<pre>package require IxTclHal ixGlobalSetDefault</pre>
SEE ALSO	

NAME - ixInitialize

ixInitialize - connects to a list of chassis, to Tcl Servers for Unix clients and opens log file.

Note: This command is deprecated in favor of the *ixConnectToChassis* and *ixConnectToTclServer* commands, which offer additional functional control.

SYNOPSIS

ixInitialize chassisList [cableLen] [/logfilename] [client]

DESCRIPTION

If this command is executed on a Unix machine or the *client* argument is “tclClient”, then *ixInitialize* establishes a TCL Server connection with the first of the chassis in *chassisList*. Use *ixConnectToTclServer* and *ixConnectToChassis* if the TCL Server is on some other host.

IxInitialize then establishes connection with IxServer running on a list of chassis and assigns chassis ID numbers to the chassis in the chain. The ID numbers are assigned in incrementing order.

In addition, it opens a log file for the script. This command should be the first one in the script file after the *package require IxTclHal*.

ARGUMENTS

chassisList

(By value) List of hostname or IP address of chassis in the chain to be connected to.

cableLen

(By value) The length of the sync cable that connects the chain of chassis
(Optional). Valid values are:

Option	Value	Usage
<i>cable3feet</i>	0	
<i>cable6feet</i>	1	default
<i>cable9feet</i>	2	
<i>cable12feet</i>	3	
<i>cable15feet</i>	4	
<i>cable18feet</i>	5	
<i>cable21feet</i>	6	
<i>cable24feet</i>	7	

logfilename

(By value) Name of the log file that is created to store all log messages while the script is running. *(Optional; default = NULL)*

client

(By value) The name of the client. *(Optional; default = local)*

RETURNS

0

No Error, connection was established with the IxServer.

1

Error connecting to IxServer; possible causes are invalid hostname or IP address for chassis, IxServer not running on the chassis, or other network problem.

- 2** Version mismatch.
- 3** Timeout connecting to chassis; possible causes are invalid hostname or IP address for chassis, or IxServer not running on the chassis.
- 5** Could not make a Tcl Server connection to the first chassis in the *chassisList*.

EXAMPLES

```
package require IxTclHal

set host1    localhost
set host2    galaxy

set ret [ixInitialize $host1]
switch $ret {
    1 {ixPuts "Error connecting to chassis"}
    2 {ixPuts "Version mismatch with chassis"}
    3 {ixPuts "Timeout connecting to chassis"}
    5 {ixPuts "Could not connect to Tcl Server"}
}
ixDisconnectFromChassis

set pl [list $host1 $host2]
set ret [ixInitialize $pl 1]
switch $ret {
    1 {ixPuts "Error connecting to chassis"}
    2 {ixPuts "Version mismatch with chassis"}
    3 {ixPuts "Timeout connecting to chassis"}
    5 {ixPuts "Could not connect to Tcl Server"}
}

ixDisconnectFromChassis
```

SEE ALSO

ixConnectToChassis, *ixDisconnectTclServer*, *ixProxyConnect*

NAME - *ixIsIntrinsicLatencyAdjustmentEnabled*

ixIsIntrinsicLatencyAdjustment Enabled - returns "true" if Intrinsic Latency is enabled

SYNOPSIS *ixIsIntrinsicLatencyAdjustmentEnabled chassisID cardID portID*

DESCRIPTION The **ixIsIntrinsicLatencyAdjustment Enabled** command returns "true" if Intrinsic Latency is enabled; otherwise returns "false".

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

RETURNS

-0 Not enabled.

-1 Enabled.

EXAMPLES *ixIsIntrinsicLatencyAdjustmentEnabled \$chassId \$cardId \$portId*

SEE ALSO *ixEnablePortIntrinsicLatencyAdjustment*
ixEnableIntrinsicLatencyAdjustment

NAME - ixIsOverlappingIpAddress

ixIsOverlappingIpAddress - compares two IP ranges for overlap

SYNOPSIS

ixIsOverlappingIpAddress *ipAddress1 count1 ipAddress2 count2*

DESCRIPTION

This command compares two IP ranges to determine whether they overlap.

ARGUMENTS

ipAddress1 The first IP address to be compared.

count1 The number of IP addresses in the first range.

ipAddress2 The second IP address to be compared.

count2 The number of IP addresses in the second range.

RETURNS

0 The ranges do not overlap.

1 The ranges overlap.

EXAMPLES

```
package require IxTclHal

set addr1    192.168.1.1
set addr1Num 300
set addr2    192.168.2.1
set addr2Num 20

if [ixIsOverlappingIpAddress $addr1 $addr1Num $addr2 $addr2Num] {
    ixPuts "Address ranges overlap"
}
```

SEE ALSO

NAME - ixIsSameSubnet

ixIsSameSubnet - compares two subnets for overlap

SYNOPSIS

ixIsSameSubnet *ipAddress1 mask1 ipAddress2 mask2*

DESCRIPTION

This command compares two subnets to determine if they are the same.

ARGUMENTS

ipAddress1 The first IP address to be compared.

mask1 The network mask for the first IP address.

ipAddress2 The second IP address to be compared.

mask2 The network mask for the first IP address.

RETURNS

0 The subnets are different.

1 The subnets are the same.

EXAMPLES

```
package require IxTclHal

set ip1 192.168.0.1
set mask1 255.255.255.0
set ip2 192.168.20.1
set mask2 255.255.0.0

if [ixIsSameSubnet $ip1 $mask1 $ip2 $mask2] {
    ixPuts "These are the same subnet"
}
```

SEE ALSO

NAME - ixIsValidHost

ixIsValidHost - determines if the host part of a masked address is valid

SYNOPSIS `ixIsValidHost ipAddress mask`

DESCRIPTION This command determines if the host part of the masked address is legal, that is, not all 0's or all 1's.

ARGUMENTS

ipAddress The IP address.

mask The network mask for the IP address.

RETURNS

0 The host part is invalid.

1 The host part is valid.

EXAMPLES

```
package require IxTclHal

set ip1 192.168.0.1
set mask1 255.255.255.0
set ip2 0.1.2.3
set mask2 255.0.0.0

if {[ixIsValidHost $ip1 $mask1] == 0} {
    ixPuts "$ip1/$mask1 is not a valid host address"
}

if {[ixIsValidHost $ip2 $mask2] == 0} {
    ixPuts "$ip2/$mask2 is not a valid host address"
}
```

SEE ALSO

NAME - ixIsValidNetMask

ixIsValidNetMask - determines if a mask is valid

SYNOPSIS *ixIsValidNetMask mask*

DESCRIPTION This command determines whether a mask is valid; that is, a set of contiguous high-order bits set, followed by a contiguous set of 0's.

ARGUMENTS

mask The network mask to be checked.

RETURNS

0 The mask is invalid.

1 The mask is valid.

EXAMPLES package require IxTclHal

```
set mask1 255.255.255.0
set mask2 0.255.0.0

if {[ixIsValidNetMask $mask1] == 0} {
    ixPuts "$mask1 is not a valid mask"
}

if {[ixIsValidNetMask $mask2] == 0} {
    ixPuts "$mask2 is not a valid mask"
}
```

SEE ALSO

NAME - ixIsValidUnicastIp

ixIsValidUnicastIp - determines if an IP address is a valid unicast address

SYNOPSIS

ixIsValidUnicastIp *ipAddress*

DESCRIPTION

This command determines whether an IP address is a valid unicast address. The address must not be 0.0.0.0 or 255.255.255.255 or 127.x.x.x or in the range 224.0.0.0 to 239.255.255.255.

ARGUMENTS

ipAddress The IP address to be checked.

RETURNS

0 The address is an invalid unicast address.

1 The address is a valid unicast address.

EXAMPLES

```
package require IxTclHal

set ip1 192.168.1.1
set ip2 240.0.0.1

if {[ixIsValidUnicastIp $ip1] == 0} {
    ixPuts "$ip1 is not a valid unicast ip"
}

if {[ixIsValidUnicastIp $ip2] == 0} {
    ixPuts "$ip2 is not a valid unicast ip"
}
```

SEE ALSO

NAME - **ixLoadPoePulse**

ixLoadPoePulse - sends a power pulse to a list of PoE powered devices

SYNOPSIS

ixLoadPoePulse portList [write]

DESCRIPTION

The **ixLoadPoePulse** command sends a pulse to a list of PoE powered device ports. All ports in the list must be for PoE load modules. The pulse parameters are set up with the *poePoweredDevice* command.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray

Or a reference to a list. For example, *pl* after

set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-

set pl {1,1,1 1,1,2 1,1,3 1,1,4}

write

(By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

SEE ALSO

ixLoadPortPoePulse

NAME - ixLoadPortPoePulse

ixLoadPortPoePulse - sends a power pulse to a PoE powered devices

SYNOPSIS

ixLoadPortPoePulse *chassisID cardID portID [write]*

DESCRIPTION

The **ixLoadPortPoePulse** command sends a pulse to a PoE powered device port. The port must be for PoE load modules. The pulse parameters are set up with the *poePoweredDevice* command.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

write (By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL
(default)

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

SEE ALSO *ixLoadPoePulse*

NAME - ixLogin

ixLogin - logs in the user

SYNOPSIS *ixLogin ixiaUser*

DESCRIPTION This command logs a user in, for purposes of port ownership.

ARGUMENTS

ixiaUser (By value) The name of the user.

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES package require IxTclHal

```
if {[ixLogin George] != 0} {  
    ixPuts "Could not log you in"  
}
```

SEE ALSO *ixLogout, ixTakeOwnership*

NAME - ixLogout

ixLogout - logs out the user

SYNOPSIS ixLogout

DESCRIPTION The **ixLogout** command logs out the user.

ARGUMENTS

None

RETURNS

- | | |
|----------|--|
| 0 | No error; the command was successfully delivered to the IxServer. |
| 1 | Error; the command was delivered to the IxServer but it could not process the message. |

EXAMPLES package require IxTclHal

```
if {[ixLogout] != 0} {  
    ixPuts "Could not log you out"  
}
```

SEE ALSO [ixLogin](#)

NAME - ixMiiConfig utilities

ixMiiConfig utilities - procedures to configure 10GE SerDes features

SYNOPSIS

```
ixMiiConfigPreEmphasis chassisID cardID portID peSetting
ixMiiConfigLossOfSignalThreshold chassisID cardID portID threshold
ixMiiConfigXgxsLinkMonitoring chassisID cardID portID enable
ixMiiConfigAlignRxDataClock chassisID cardID portID clock
ixMiiConfigReceiveEqualization chassisID cardID portID equalization
ixMiiConfigXauiOutput chassisID cardID portID enable
ixMiiConfigXauiSerialLoopback chassisID cardID portID enable
ixMiiConfigXgmiiParallelLoopback chassisID cardID portID enable
```

DESCRIPTION

The following procedures configure the MII registers for 10GE modules:

- **ixMiiConfigPreEmphasis**: configures output pre-emphasis.
- **ixMiiConfigLossOfSignalThreshold**: configures the receive loss of signal threshold.
- **ixMiiConfigXgxsLinkMonitoring**: enables or disables link monitoring.
- **ixMiiConfigAlignRxDataClock**: aligns the receive clock with the recovered clock or internal reference clock.
- **ixMiiConfigReceiveEqualization**: sets the value of the receive equalization.
- **ixMiiConfigXauiOutput**: for XAUI modules, enables output.
- **ixMiiConfigXauiSerialLoopback**: for XAUI modules, enables serial loopback.
- **ixMiiConfigXgmiiParallelLoopback**: for XAUI modules, enables parallel loopback.

ARGUMENTS

cardID

(By value) The ID number of the card.

chassisID

(By value) The ID number of the chassis.

clock

(By value) For use with **ixMiiConfigAlignRxDataClock**, set the receive clock alignment. One of:

Option	Value	Usage
<i>miiRecoveredClock</i>	0	Use the recovered clock.
<i>miiLocalRefClock</i>	1	Use the local reference clock.

enable true / false

(By value) For use with **ixMiiConfigXgxsLinkMonitoring**, **ixMiiConfigXauiOutput**, **ixMiiConfigXauiSerialLoopback** or **ixMiiConfigXgmiiParallelLoopback**. Enables or disables the feature.

equalizationValue (By value) For use with **ixMiiConfigReceiveEqualization**, the receive equalization value between 0 and 15.

portID (By value) The ID number of the port.

peSetting (By value) For use with **ixMiiConfigPreEmphasis**, the pre-emphasis setting. One of:

Option	Value	Usage
<i>miiPreemphasisNone</i>	0	No pre-emphasis.
<i>miiPreemphasis18</i>	1 or 18	A value of 18%.
<i>miiPreemphasis38</i>	2 or 38	A value of 38%.
<i>miiPreemphasis75</i>	3 or 75	A value of 75%.

threshold (By value) For use with **ixMiiConfigLossOfSignalThreshold**, the loss of signal threshold setting. One of:

Option	Value	Usage
<i>miiLossOfSignal160mv</i>	0 or 160	A value of 160mv.
<i>miiLossOfSignal240mv</i>	1 or 240	A value of 240mv.
<i>miiLossOfSignal200mv</i>	2 or 200	A value of 200mv.
<i>miiLossOfSignal120mv</i>	3 or 120	A value of 120mv.
<i>miiLossOfSignal80mv</i>	4 or 80	A value of 80mv.

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error.

EXAMPLES

SEE ALSO

mii

NAME - ixPortClearOwnership

ixPortClearOwnership - clears ownership of a single port

SYNOPSIS

ixPortClearOwnership *chassisID cardID portID [takeType]*

DESCRIPTION

The **ixPortClearOwnership** command clears ownership of the specified port.

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.
takeType	(By value) (<i>Optional</i>) Valid values: <i>force</i> - take regardless of whether the port is owned by someone else

RETURNS

0	No error; the command was successfully delivered to the IxServer.
1	Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package req IxTclHal

set host galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

set chas [ixGetChassisID $host]
set card 1
set port 1

if {[ixPortClearOwnership $chas $card $port] != 0} {
    ixPuts "Could not clear ownership for $chas:$card$port"
}
```

```
if {[ixPortClearOwnership $chas $card $port force] != 0} {
    ixPuts "Could not clear ownership for $chas:$card$port"
}

# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixPortTakeOwnership, *ixClearOwnership*, *ixTakeOwnership*

NAME - ixPortTakeOwnership

ixPortTakeOwnership - takes ownership of a single port

SYNOPSIS

ixPortTakeOwnership *chassisID cardID portID [takeType]*

DESCRIPTION

The **ixPortTakeOwnership** command takes ownership of the specified port.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

takeType (By value) (*Optional*) Valid values:

force - take regardless of whether the port is owned by someone else

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLE

```
package req IxTclHal

set host galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

set chas [ixGetChassisID $host]
set card 1
set port 1

if {[ixPortTakeOwnership $chas $card $port] != 0} {
    ixPuts "Could not Take ownership for $chas:$card$port"
}
```

```
if {[ixPortTakeOwnership $chas $card $port force] != 0} {  
    ixPuts "Could not Take ownership for $chas:$card$port"  
}  
  
# Disconnect from the chassis we're using  
ixDisconnectFromChassis $host  
# If we're running on UNIX, disconnect from the TCL Server  
if [isUNIX] {  
    ixDisconnectTclServer $host  
}
```

SEE ALSO

ixPortClearOwnership, *ixClearOwnership*, *ixTakeOwnership*

NAME - ixProxyConnect

ixProxyConnect - connects to a list of chassis, to Tcl Servers for Unix clients and opens log file

SYNOPSIS

ixProxyConnect *tclSrv chassisList [cableLen] [logfilename]*

DESCRIPTION

The **ixProxyConnect** command establishes connection with IxServer running on a list of chassis and assigns chassis ID numbers to the chassis in the chain. The ID numbers are assigned in incrementing order to the master and slave chassis with the master chassis given ID 1. The command also connects to the Tcl Server on the specified host. Also, it opens a log file for the script.

ARGUMENTS

tclSrv

(By value) The hostname of the computer running the TclServer.

chassisList

(By value) List of hostname or IP address of chassis in the chain to be connected to.

cableLen

(By value) The length of the sync cable that connects the chain of chassis
(Optional). Valid values are:

Option	Value	Usage
<i>cable3feet</i>	0	default
<i>cable6feet</i>	1	
<i>cable9feet</i>	2	
<i>cable12feet</i>	3	
<i>cable15feet</i>	4	
<i>cable18feet</i>	5	
<i>cable21feet</i>	6	
<i>cable24feet</i>	7	

logfilename

(By value) Name of the log file that is created to store all log messages while the script is running. *(Optional; default = NULL)*

RETURNS

0

No Error, connection was established with the IxServer.

1

Error connecting to IxServer; possible causes are invalid hostname or IP address for chassis, IxServer not running on the chassis, or other network problem.

2

Version mismatch.

3

Timeout connecting to chassis; possible causes are invalid hostname or IP address for chassis, or IxServer not running on the chassis.

5

Could not make a Tcl Server connection to *tclSrv*.

EXAMPLES

```
package require IxTclHal

set host1      localhost
set host2      galaxy
set tclServer  galaxy

set ret [ixProxyConnect $tclServer $host1]
switch $ret {
    1 {ixPuts "Error connecting to chassis"}
    2 {ixPuts "Version mismatch with chassis"}
    3 {ixPuts "Timeout connecting to chassis"}
    5 {ixPuts "Could not connect to Tcl Server"}
}
ixDisconnectFromChassis
ixDisconnectTclServer

set pl [list $host1 $host2]
set ret [ixProxyConnect $tclServer $pl $::cable6feet]
switch $ret {
    1 {ixPuts "Error connecting to chassis"}
    2 {ixPuts "Version mismatch with chassis"}
    3 {ixPuts "Timeout connecting to chassis"}
    5 {ixPuts "Could not connect to Tcl Server"}
}

ixDisconnectFromChassis
ixDisconnectTclServer
```

SEE ALSO

ixConnectToChassis, *ixConnectToTclServer*, *ixDisconnectTclServer*

NAME - ixPuts

ixPuts - output text to the console.

SYNOPSIS *ixPuts [-nonewline] arg...*

DESCRIPTION The **ixPuts** command outputs its arguments to the console window with or without a newline.

ARGUMENTS

-nonewline If present, suppresses a newline at the end of the output.

arg ... Arguments which are concatenated together and displayed on the console.

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLE package require IxTclHal

```
ixPuts "hello"  
ixPuts -nonewline "This will "  
ixPuts -nonewline "all be displayed "  
ixPuts "on the same line"
```

SEE ALSO *logMsg*

NAME - ixRequestStats

ixRequestStats - request statistics for a group of ports

SYNOPSIS ixRequestStats *portList*

DESCRIPTION The *ixRequestStats* command requests that the statistics for a group of ports be retrieved. The statistics may be read through the use of the *statList* command.

ARGUMENTS

portList (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0 Statistics were retrieved.

1 An error occurred.

EXAMPLES

SEE ALSO [statList](#)

NAME - ixResetPortSequenceIndex

ixResetPortSequenceIndex - reset a port's sequence index

SYNOPSIS	ixResetPortSequenceIndex <i>chassisID cardID portID</i>
-----------------	---

DESCRIPTION	The ixResetPortSequenceIndex command sends a message to the IxServer to reset the sequence number associated with a port.
--------------------	--

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.

RETURNS

0	No error; the command was successfully delivered to the IxServer
1	Error; the command was delivered to the IxServer but it could not process the message

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA 1
set portA 1
set portList [list [list $chas $cardA $portA]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

if {[ixResetPortSequenceIndex $chas $cardA $portA] != 0} {
    ixPuts "Could not reset port sequence index for
$chas:$cardA:$portA"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixResetSequenceIndex

NAME - ixResetSequenceIndex

ixResetSequenceIndex - reset a group of ports' sequence index

SYNOPSIS ixResetSequenceIndex *portList*

DESCRIPTION The **ixResetSequenceIndex** command sends a message to the IxServer to reset the sequence index associated with a group of ports.

ARGUMENTS

portList (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set chas    1
set cardA   1
set portA   1
set cardB   1
set portB   2

# Four different port list formats
```

```

set portList1 [list $chas,$cardA,$portA]
set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chas $cardA $portA      $chas $cardB $portB
map add      $chas $cardB $portB      $chas $cardA $portA

# Try each of the formats
if {[ixResetSequenceIndex portList1] != 0} {
    ixPuts "Could not reset sequence index for $portList1"
}
if {[ixResetSequenceIndex portList2] != 0} {
    ixPuts "Could not reset sequence index for $portList2"
}
if {[ixResetSequenceIndex portList3] != 0} {
    ixPuts "Could not reset sequence index for $portList3"
}
if {[ixResetSequenceIndex portList4] != 0} {
    ixPuts "Could not reset sequence index for $portList4"
}
if {[ixResetSequenceIndex one2oneArray] != 0} {
    ixPuts "Could not reset sequence index for $one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixResetPortSequenceIndex*

NAME - *ixRestartAutoNegotiation*

ixRestartAutoNegotiation - restart auto-negotiation on a set of ports

SYNOPSIS *ixRestartAutoNegotiation portList*

DESCRIPTION The **ixRestartAutoNegotiation** command sends a message to the IxServer to restart the auto-negotiation on a group of ports.

ARGUMENTS

portList (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
set portList {{1 1 1} {1 1 2}}
if [ixRestartAutoNegotiation portList] {
    ixPuts $::ixErrorInfo
}
```

SEE ALSO *ixRestartPortAutoNegotiation*

NAME - ixRestartPortAutoNegotiation

ixRestartPortAutoNegotiation - restart auto-negotiation on a port

SYNOPSIS

ixRestartPortAutoNegotiation *chassisID cardID portID*

DESCRIPTION

The **ixRestartPortAutoNegotiation** command sends a message to the IxServer to restart the auto-negotiation on port.

ARGUMENTS

- chassisID** (By value) The ID number of the chassis.
cardID (By value) The ID number of the card.
portID (By value) The ID number of the port.

RETURNS

- 0** No error; the command was successfully delivered to the IxServer.
1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
if [ixRestartPortAutoNegotiation 1 1 1] {  
    ixPuts $::ixErrorInfo  
}
```

SEE ALSO

ixRestartAutoNegotiation

NAME - ixRestartPortPPPAutoNegotiation

ixRestartPortPPPAutoNegotiation - restart PPP negotiation on a port

SYNOPSIS

ixRestartPortPPPNegotiation chassisID cardID portID

DESCRIPTION

The **ixRestartPortPPPAutoNegotiation** command sends a message to the IxServer to restart the PPP negotiation on port.

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.

RETURNS

0	No error; the command was successfully delivered to the IxServer.
1	Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
if [ixRestartPortPPPNegotiation 1 1 1] {  
    ixPuts $::ixErrorInfo  
}
```

SEE ALSO

ixRestartPPPNegotiation

NAME - ixRestartPPPNegotiation

ixRestartPPPNegotiation - restart PPP negotiation on a set of ports

SYNOPSIS

ixRestartPPPNegotiation *portList*

DESCRIPTION

The **ixRestartPPPNegotiation** command sends a message to the IxServer to restart the PPP negotiation on a group of ports.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
set portList {{1 1 1} {1 1 2}}
if [ixRestartPPPNegotiation portList] {
    ixPuts $::ixErrorInfo
}
```

SEE ALSO

ixRestartPPPNegotiation

NAME - ixSetAdvancedStreamSchedulerMode

ixSetAdvancedStreamSchedulerMode - set a group of ports to advanced stream scheduler transmit mode

SYNOPSIS

ixSetAdvancedStreamSchedulerMode *portList* [*write*]

DESCRIPTION

The **ixSetAdvancedStreamSchedulerMode** command sends a message to the IxServer to set the transmit mode of a group of ports simultaneously to advanced stream scheduler mode. The ports may span multiple chassis.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

write

(By value) (*Optional*) Valid values:

write: the action is committed to hardware
noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host    localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
```

```
set chas [ixGetChassisID $host]

set chas    1
set cardA   4
set portA   1
set cardB   4
set portB   2

set pl1 [list 1,$cardA,$portA]
set pl2 [list 1,$cardA,$portA 1,$cardB,$portB]
set pl3 [list [list $chas $cardA $portA] [list 1 $cardB $portB]]
set pl4 [list [list 1,$cardA,$portA] [list 1,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $pl4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      1 $cardA $portA   1 $cardB $portB
map add      1 $cardB $portB   1 $cardA $portA

if {[ixSetAdvancedStreamSchedulerMode pl1 write] != 0} {
    puts "Could not set advanced stream scheduler mode for $pl1"
}
if {[ixSetAdvancedStreamSchedulerMode pl2 write] != 0} {
    puts "Could not set advanced stream scheduler mode for $pl2"
}
if {[ixSetAdvancedStreamSchedulerMode pl3 write] != 0} {
    puts "Could not set advanced stream scheduler mode for $pl3"
}
if {[ixSetAdvancedStreamSchedulerMode pl4 write] != 0} {
    puts "Could not set advanced stream scheduler mode for $pl4"
}
if {[ixSetAdvancedStreamSchedulerMode oneZoneArray write] != 0} {
    puts "Could not set advanced stream scheduler mode for $oneZoneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $pl4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixSetPortAdvancedStreamSchedulerMode

NAME - *ixSetAutoDetectInstrumentationMode*

ixSetAutoDetectInstrumentationMode - This command sets all the RX ports in the list or array to all the auto instrumentation modes, that is, Packet Groups, Data Integrity, and Sequence Checking

SYNOPSIS

ixSetAutoDetectInstrumentationMode portList [write]

DESCRIPTION

This command allows the receive side of a port to trigger on a set pattern, that can be matched in the packet. The port looks in Packet Groups, Data Integrity, and Sequence Checking headers, as well as start at a specific offset (if configured).

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

write

(By value) (*Optional*) Valid values:

write: the action is committed to hardware
noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

SEE ALSO

NAME - ixSetCaptureMode

ixSetCaptureMode - set a group of ports to Capture Receive mode

SYNOPSIS

ixSetCaptureMode *portList* [*write*]

DESCRIPTION

The **ixSetCaptureMode** command sends a message to the IxServer to set the receive mode of a group of ports simultaneously to Capture mode. The ports may span multiple chassis. This mode must be used when traffic is to be captured in the capture buffer. This mode is mutually exclusive with the Packet Group receive mode.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

write

(By value) (*Optional*) Valid values:

write: the action is committed to hardware
noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set chas    1
set cardA   1
set portA   1
set cardB   1
set portB   2

set pl1 [list 1,$cardA,$portA]
set pl2 [list 1,$cardA,$portA 1,$cardB,$portB]
set pl3 [list [list $chas $cardA $portA] [list 1 $cardB $portB]]
set pl4 [list [list 1,$cardA,$portA] [list 1,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $pl4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type oneZone
map config -type oneZone
map add      1 $cardA $portA   1 $cardB $portB
map add      1 $cardB $portB   1 $cardA $portA

if {[ixSetCaptureMode pl1 write] != 0} {
    puts "Could not set capture mode for $pl1"
}
if {[ixSetCaptureMode pl2 write] != 0} {
    puts "Could not set capture mode for $pl2"
}
if {[ixSetCaptureMode pl3 write] != 0} {
    puts "Could not set capture mode for $pl3"
}
if {[ixSetCaptureMode pl4 write] != 0} {
    puts "Could not set capture mode for $pl4"
}
if {[ixSetCaptureMode oneZoneArray write] != 0} {
    puts "Could not set capture mode for $oneZoneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $pl4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixSetPortCaptureMode*

NAME - ixSetDataIntegrityMode

ixSetDataIntegrityMode - set a group of ports to Data Integrity Receive mode

SYNOPSIS

ixSetDataIntegrityMode *portList* [*write*]

DESCRIPTION

The **ixSetDataIntegrityMode** command sends a message to the IxServer to set the receive mode of a group of ports simultaneously to Data Integrity mode. The ports may span multiple chassis.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

write

(By value) (*Optional*) Valid values:

write: the action is committed to hardware
noWrite: the action is not committed to hardware but just set in IxHAL
(default)

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
```

```

set cardA 4
set portA 1
set cardB 4
set portB 2

set pl1 [list 1,$cardA,$portA]
set pl2 [list 1,$cardA,$portA 1,$cardB,$portB]
set pl3 [list [list $chas $cardA $portA] [list 1 $cardB $portB]]
set pl4 [list [list 1,$cardA,$portA] [list 1,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $pl4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2zone
map config -type one2zone
map add      1 $cardA $portA   1 $cardB $portB
map add      1 $cardB $portB   1 $cardA $portA

if {[ixSetDataIntegrityMode pl1 write] != 0} {
    puts "Could not set data integrity mode for $pl1"
}
if {[ixSetDataIntegrityMode pl2 write] != 0} {
    puts "Could not set data integrity mode for $pl2"
}
if {[ixSetDataIntegrityMode pl3 write] != 0} {
    puts "Could not set data integrity mode for $pl3"
}
if {[ixSetDataIntegrityMode pl4 write] != 0} {
    puts "Could not set data integrity mode for $pl4"
}
if {[ixSetDataIntegrityMode one2oneArray write] != 0} {
    puts "Could not set data integrity mode for $one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $pl4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixSetPortDataIntegrityMode*

NAME - ixSetPacketFlowMode

ixSetPacketFlowMode - set a group of ports to Packet Flow Transmit mode

SYNOPSIS

ixSetPacketFlowMode *portList* [*write*]

DESCRIPTION

The **ixSetPacketFlowMode** command sends a message to the IxServer to set the transmit mode of a group of ports simultaneously to Packet Flow mode. The ports may span multiple chassis. This mode is mutually exclusive with the Packet Streams transmit mode.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray

Or a reference to a list. For example, *pl* after

set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-

set pl {1,1,1 1,1,2 1,1,3 1,1,4}

write

(By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL

(default)

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
```

```

set chas [ixGetChassisID $host]

set cardA 1
set portA 1
set cardB 1
set portB 2

set pl1 [list 1,$cardA,$portA]
set pl2 [list 1,$cardA,$portA 1,$cardB,$portB]
set pl3 [list [list $chas $cardA $portA] [list 1 $cardB $portB]]
set pl4 [list [list 1,$cardA,$portA] [list 1,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $pl4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add 1 $cardA $portA 1 $cardB $portB
map add 1 $cardB $portB 1 $cardA $portA

if {[ixSetPacketFlowMode pl1 write] != 0} {
    puts "Could not set PacketFlow mode for $pl1"
}
if {[ixSetPacketFlowMode pl2 write] != 0} {
    puts "Could not set PacketFlow mode for $pl2"
}
if {[ixSetPacketFlowMode pl3 write] != 0} {
    puts "Could not set PacketFlow mode for $pl3"
}
if {[ixSetPacketFlowMode pl4 write] != 0} {
    puts "Could not set PacketFlow mode for $pl4"
}
if {[ixSetPacketFlowMode one2oneArray write] != 0} {
    puts "Could not set PacketFlow mode for $one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $pl4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

ixSetPortPacketFlowMode, *ixSetPacketStreamMode*,
ixSetPortPacketStreamMode

NAME - ixSetPacketGroupMode

ixSetPacketGroupMode - set a group of ports to Packet Group Receive mode

SYNOPSIS

ixSetPacketGroupMode *portList* [*write*]

DESCRIPTION

The **ixSetPacketGroupMode** command sends a message to the IxServer to set the receive mode of a group of ports simultaneously to Packet Group mode. The ports may span multiple chassis. This mode must be used when real-time latency metrics are to be obtained.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

write

(By value) (*Optional*) Valid values:

write: the action is committed to hardware
noWrite: the action is not committed to hardware but just set in IxHAL
(default)

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
set cardA 1
```

```

set portA    1
set cardB   1
set portB   2

set pl1 [list 1,$cardA,$portA]
set pl2 [list 1,$cardA,$portA 1,$cardB,$portB]
set pl3 [list [list $chas $cardA $portA] [list 1 $cardB $portB]]
set pl4 [list [list 1,$cardA,$portA] [list 1,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $pl4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type onezone
map add      1 $cardA $portA   1 $cardB $portB
map add      1 $cardB $portB   1 $cardA $portA

if {[ixSetPacketGroupMode pl1 write] != 0} {
    puts "Could not set PacketGroup mode for $pl1"
}

map new -type one2one
map config -type onezone
map add      1 $cardA $portA   1 $cardB $portB
map add      1 $cardB $portB   1 $cardA $portA

if {[ixSetPacketGroupMode pl1 write] != 0} {
    puts "Could not set PacketGroup mode for $pl1"
}
if {[ixSetPacketGroupMode pl2 write] != 0} {
    puts "Could not set PacketGroup mode for $pl2"
}
if {[ixSetPacketGroupMode pl3 write] != 0} {
    puts "Could not set PacketGroup mode for $pl3"
}
if {[ixSetPacketGroupMode pl4 write] != 0} {
    puts "Could not set PacketGroup mode for $pl4"
}
if {[ixSetPacketGroupMode oneZoneArray write] != 0} {
    puts "Could not set PacketGroup mode for $oneZoneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $pl4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixSetPortPacketGroupMode*

NAME - ixSetPacketStreamMode

ixSetPacketStreamMode - set a group of ports to Packet Stream Transmit mode

SYNOPSIS

ixSetPacketStreamMode *portList [write]*

DESCRIPTION

The **ixSetPacketStreamMode** command sends a message to the IxServer to set the transmit mode of a group of ports simultaneously to Packet Stream mode. The ports may span multiple chassis. This mode is mutually exclusive with the Packet Flow transmit mode.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

write

(By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
```

```

set chas [ixGetChassisID $host]

set chas    1
set cardA   1
set portA   1
set cardB   1
set portB   2

set pl1 [list 1,$cardA,$portA]
set pl2 [list 1,$cardA,$portA 1,$cardB,$portB]
set pl3 [list [list $chas $cardA $portA] [list 1 $cardB $portB]]
set pl4 [list [list 1,$cardA,$portA] [list 1,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $pl4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      1 $cardA $portA   1 $cardB $portB
map add      1 $cardB $portB   1 $cardA $portA

if {[ixSetPacketStreamMode pl1 write] != 0} {
    puts "Could not set PacketStream mode for $pl1"
}
if {[ixSetPacketStreamMode pl2 write] != 0} {
    puts "Could not set PacketStream mode for $pl2"
}
if {[ixSetPacketStreamMode pl3 write] != 0} {
    puts "Could not set PacketStream mode for $pl3"
}
if {[ixSetPacketStreamMode pl4 write] != 0} {
    puts "Could not set PacketStream mode for $pl4"
}
if {[ixSetPacketStreamMode one2oneArray write] != 0} {
    puts "Could not set PacketStream mode for $one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $pl4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixSetPortPacketStreamMode*

NAME - ixSetPortAdvancedStreamSchedulerMode

ixSetPortAdvancedStreamSchedulerMode - set a port to advanced stream scheduler transmit mode

SYNOPSIS

ixSetPortAdvancedStreamSchedulerMode *chassisID cardID portID [write]*

DESCRIPTION

The **ixSetPortAdvancedStreamSchedulerMode** command sends a message to the IxServer to set the transmit mode of a single port to advanced stream scheduler transmit mode.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

write (By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
```

```
set card    1
set port    1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

if {[ixSetPortAdvancedStreamSchedulerMode $chas $card $port write]
!= 0} {
    ixPuts "Could not set port $chas:$card:$port to advanced
stream scheduler mode"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*ixSetAdvancedStreamSchedulerMode*

NAME - ixSetPortCaptureMode

ixSetPortCaptureMode - set a port to capture mode

SYNOPSIS

ixSetPortCaptureMode *chassisID cardID portID [write]*

DESCRIPTION

The **ixSetPortCaptureMode** command sends a message to the IxServer to set the receive mode of a single port to Capture mode. This mode must be used when traffic is to be captured in the capture buffer. This mode is mutually exclusive with the Packet Group receive mode.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

write (By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL

(*default*)

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
```

```
set card    1
set port    1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

if {[ixSetPortCaptureMode $chas $card $port write] != 0} {
    ixPuts "Could not set port $chas:$card:$port to capture mode"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*ixSetCaptureMode*

NAME - ixSetPortDataIntegrityMode

ixSetPortDataIntegrityMode - set a port to data integrity mode

SYNOPSIS

ixSetPortDataIntegrityMode *chassisID cardID portID [write]*

DESCRIPTION

The **ixSetPortDataIntegrityMode** command sends a message to the IxServer to set the receive mode of a single port to Data Integrity mode.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

write (By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card     1
```

```
set port      1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

if {[ixSetPortDataIntegrityMode $chas $card $port write] != 0} {
    ixPuts "Could not set port $chas:$card:$port to data integrity
mode"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*ixSetDataIntegrityMode*

NAME - ixSetPortPacketFlowMode

ixSetPortPacketFlowMode - set a port to Packet Flow Transmit mode

SYNOPSIS

ixSetPortPacketFlowMode *chassisID cardID portID [write]*

DESCRIPTION

The **ixSetPortPacketFlowMode** command sends a message to the IxServer to set the transmit mode of a single port to Packet Flow mode. This mode is mutually exclusive with the Packet Streams transmit mode.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

write (By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0 No error; the command was successfully delivered to the IxServer

1 Error; the command was delivered to the IxServer but it could not process the message

EXAMPLES

```
package require IxTclHal

set host      galaxy

set chas      1
set card      1
set port      1

package require IxTclHal

set host      localhost
set username  user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
```

```
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card    1
set port    1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

if {[ixSetPortPacketFlowMode $chas $card $port write] != 0} {
    ixPuts "Could not set port $chas:$card:$port to PacketFlow
mode"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*ixSetPacketFlowMode*

NAME - ixSetPortPacketGroupMode

ixSetPortPacketGroupMode - set a port to Packet Group Receive mode

SYNOPSIS

ixSetPortPacketGroupMode *chassisID cardID portID [write]*

DESCRIPTION

The **ixSetPortPacketGroupMode** command sends a message to the IxServer to set the receive mode of a single port to Packet Group mode. This mode must be used when real-time latency metrics are to be obtained.

ARGUMENTS

chassisID (By value) The ID number of the chassis

cardID (By value) The ID number of the card

portID (By value) The ID number of the port

write (By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      galaxy

set chas      1
set card      1
set port      1

package require IxTclHal

set host      localhost
set username  user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
```

```
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card    1
set port    1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

if {[ixSetPortPacketGroupMode $chas $card $port write] != 0} {
    ixPuts "Could not set port $chas:$card:$port to PacketGroup
mode"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*ixSetPortPacketGroupMode*

NAME - ixSetPortPacketStreamMode

ixSetPortPacketStreamMode - set a port to Packet Stream Transmit mode

SYNOPSIS

ixSetPortPacketStreamMode *chassisID cardID portID [write]*

DESCRIPTION

The **ixSetPortPacketStreamMode** command sends a message to the IxServer to set the transmit mode of a single port to Packet Stream mode. This mode is mutually exclusive with the Packet Flow transmit mode.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

write (By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
```

```
set card      1
set port      1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

if {[ixSetPortPacketStreamMode $chas $card $port write] != 0} {
    ixPuts "Could not set port $chas:$card:$port to PacketStream
mode"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixSetPacketStreamMode, *ixSetPacketFlowMode*, *ixSetPortPacketFlowMode*

NAME - ixSetPortSequenceCheckingMode

ixSetPortSequenceCheckingMode - set a port to sequence checking mode

SYNOPSIS

ixSetPortSequenceCheckingMode *chassisID cardID portID [write]*

DESCRIPTION

The **ixSetPortSequenceCheckingMode** command sends a message to the IxServer to set the receive mode of a single port to sequence checking mode.

ARGUMENTS

chassisID

(By value) The ID number of the chassis.

cardID

(By value) The ID number of the card.

portID

(By value) The ID number of the port.

write

(By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card     1
```

```
set port      1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

if {[ixSetPortSequenceCheckingMode $chas $card $port write] != 0}
{
    ixPuts "Could not set port $chas:$card:$port to sequence
           checking mode"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*ixSetSequenceCheckingMode*

NAME - ixSetPortTcpRoundTripFlowMode

ixSetPortTcpRoundTripFlowMode - set a port to TCP Round Trip Flow mode

SYNOPSIS

ixSetPortTcpRoundTripFlowMode *chassisID cardID portID [write]*

DESCRIPTION

The **ixSetPortTcpRoundTripFlowMode** command sends a message to the IxServer to set the transmit mode of a single port to TCP Round Trip Flow mode.

ARGUMENTS

chassisID (By value) The ID number of the chassis.

cardID (By value) The ID number of the card.

portID (By value) The ID number of the port.

write (By value) (*Optional*) Valid values:

write: the action is committed to hardware

noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      localhost
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
```

```
set card      1
set port      1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

if {[ixSetPortTcpRoundTripFlowMode $chas $card $port write] != 0}
{
    ixPuts "Could not set port $chas:$card:$port to
TcpRoundTripFlow mode"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*ixSetTcpRoundTripFlowMode*

NAME - ::ixSetScheduledTransmitTime

ixSetScheduledTransmitTime - set the amount of transmit time for a port list

SYNOPSIS

`ixSetScheduledTransmitTime portList duration`

DESCRIPTION

Sets the maximum amount of time that a group of ports transmits. This is only valid for ports that support the *portFeatureScheduledTxDuration* feature, which may be tested with the *port isValidFeature* command.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
`set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}}` -or-
`set pl {1,1,1 1,1,2 1,1,3 1,1,4}`

duration

(By value) The duration, in seconds, of the transmit time.

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
set portList {{1 1 1} {1 1 2}}
if [ixSetScheduledTransmitTime portList 20] {
    ixPuts $::ixErrorInfo
}
```

SEE ALSO

[ixClearScheduledTransmitTime](#)

NAME - ixSetSequenceCheckingMode

ixSetSequenceCheckingMode - set a group of ports to Sequence Checking Receive mode

SYNOPSIS	ixSetSequenceCheckingMode <i>portList</i> [<i>write</i>]
-----------------	--

DESCRIPTION	The ixSetSequenceCheckingMode command sends a message to the IxServer to set the receive mode of a group of ports simultaneously to Sequence Checking mode. The ports may span multiple chassis.
--------------------	--

ARGUMENTS

portList	(By reference) The list of ports in one of the following formats:
-----------------	---

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,2 1,1,3 1,1,4}

write	(By value) (<i>Optional</i>) Valid values:
--------------	--

write: the action is committed to hardware
noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0	No error; the command was successfully delivered to the IxServer.
----------	---

1	Error; the command was delivered to the IxServer but it could not process the message.
----------	--

EXAMPLES

```
package require IxTclHal

set host    localhost
set username user

# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
set cardA 4
set portA 1
set cardB 4
set portB 2

set pl1 [list 1,$cardA,$portA]
set pl2 [list 1,$cardA,$portA 1,$cardB,$portB]
set pl3 [list [list $chas $cardA $portA] [list 1 $cardB $portB]]
set pl4 [list [list 1,$cardA,$portA] [list 1,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $pl4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      1 $cardA $portA   1 $cardB $portB
map add      1 $cardB $portB   1 $cardA $portA

if {[ixSetSequenceCheckingMode pl1 write] != 0} {
    puts "Could not set sequence checking mode for $pl1"
}
if {[ixSetSequenceCheckingMode pl2 write] != 0} {
    puts "Could not set sequence checking mode for $pl2"
}
if {[ixSetSequenceCheckingMode pl3 write] != 0} {
    puts "Could not set sequence checking mode for $pl3"
}
if {[ixSetSequenceCheckingMode pl4 write] != 0} {
    puts "Could not set sequence checking mode for $pl4"
}
if {[ixSetSequenceCheckingMode one2oneArray write] != 0} {
    puts "Could not set sequence checking mode for $one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $pl4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixSetPortSequenceCheckingMode*

NAME - ixSetTcpRoundTripFlowMode

ixSetTcpRoundTripFlowMode - set a group of ports to TCP Round Trip Flow mode

SYNOPSIS	ixSetTcpRoundTripFlowMode <i>portList</i> [<i>write</i>]
-----------------	--

DESCRIPTION	The ixSetTcpRoundTripFlowMode command sends a message to the IxServer to set the flow mode of a group of ports simultaneously to TCP Round Trip mode. The ports may span multiple chassis.
--------------------	---

ARGUMENTS

portList	(By reference) The list of ports in one of the following formats:
-----------------	---

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

write	(By value) (<i>Optional</i>) Valid values:
--------------	--

write: the action is committed to hardware
noWrite: the action is not committed to hardware but just set in IxHAL
(*default*)

RETURNS

0	No error; the command was successfully delivered to the IxServer.
----------	---

1	Error; the command was delivered to the IxServer but it could not process the message.
----------	--

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
```

```

set chas [ixGetChassisID $host]

set cardA 1
set portA 1
set cardB 1
set portB 2

set pl1 [list 1,$cardA,$portA]
set pl2 [list 1,$cardA,$portA 1,$cardB,$portB]
set pl3 [list [list $chas $cardA $portA] [list 1 $cardB $portB]]
set pl4 [list [list 1,$cardA,$portA] [list 1,$cardB,$portB]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $pl4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      1 $cardA $portA  1 $cardB $portB
map add      1 $cardB $portB  1 $cardA $portA

if {[ixSetTcpRoundTripFlowMode pl1 write] != 0} {
    puts "Could not set PacketFlowMode for $pl1"
}
if {[ixSetTcpRoundTripFlowMode pl2 write] != 0} {
    puts "Could not set PacketFlowMode for $pl2"
}
if {[ixSetTcpRoundTripFlowMode pl3 write] != 0} {
    puts "Could not set PacketFlowMode for $pl3"
}
if {[ixSetTcpRoundTripFlowMode pl4 write] != 0} {
    puts "Could not set PacketFlowMode for $pl4"
}
if {[ixSetTcpRoundTripFlowMode one2oneArray write] != 0} {
    puts "Could not set PacketFlowMode for $one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $pl4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

[ixSetPortTcpRoundTripFlowMode](#)

NAME - For example,: : **ixSimulatePhysicalInterfaceDown**

ixSimulatePhysicalInterfaceDown - This command simulates physical interface down on a port list

SYNOPSIS ixSimulatePhysicalInterfaceDown *TxRxArray*

DESCRIPTION The **ixSimulatePhysicalInterfaceDown** command simulates that the status of physical interfaces in a port list is down.

ARGUMENTS

TxRxArray Either list of ports or array of ports.

RETURNS

Code The return code from *simulatePhysicalInterfaceDown*.

EXAMPLES

```
proc ixSimulatePhysicalInterfaceDown {TxRxArray} \
{
    upvar $TxRxArray txRxArray
    return [simulatePhysicalInterfaceDown txRxArray]
}
```

SEE ALSO

NAME - ixSimulatePhysicalInterfaceUp

ixSimulatePhysicalInterfaceUp - This command simulates physical interface up on a port list

SYNOPSIS

ixSimulatePhysicalInterfaceUp *TxRxArray*

DESCRIPTION

The **ixSimulatePhysicalInterfaceUp** command simulates that the status of physical interfaces in a port list is up.

ARGUMENTS

TxRxArray

Either list of ports or array of ports.

RETURNS

Code

The return code from *simulatePhysicalInterfaceUp*.

EXAMPLES

```
proc ixSimulatePhysicalInterfaceUp {TxRxArray} \
{
    upvar $TxRxArray      txRxArray
    return [simulatePhysicalInterfaceUp txRxArray]
}
```

SEE ALSO

NAME - ixSimulatePortPhysicalInterfaceDown

ixSimulatePortPhysicalInterfaceDown - This command simulates physical interface down on a single port.

SYNOPSIS *ixSimulatePortPhysicalInterfaceDown chassisID cardID portID*

DESCRIPTION The **ixSimulatePortPhysicalInterfaceDown** command simulates that the status of single physical port is down.

ARGUMENTS**chassisID** (By value) The ID number of the chassis.**cardID** (By value) The ID number of the card.**portID** (By value) The ID number of the port.

RETURNS**Code** The return code from *simulatePortPhysicalInterfaceDown*.

EXAMPLES

```
set retCode $::TCL_OK
return [ixSimulatePortPhysicalInterfaceDown $chassis $lm $port]
```

SEE ALSO

NAME - ixSimulatePortPhysicalInterfaceUp

ixSimulatePortPhysicalInterfaceUp - This command simulates physical interface up on a single port.

SYNOPSIS

ixSimulatePortPhysicalInterfaceUp *chassisID cardID portID*

DESCRIPTION

The **ixSimulatePortPhysicalInterfaceUp** command simulates that the status of single physical port is Up.

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.

RETURNS

Code	The return code from <i>simulatePortPhysicalInterfaceUp</i> .
-------------	---

EXAMPLES

```
set retCode $::TCL_OK  
return [ixSimulatePortPhysicalInterfaceUp $chassis $lm $port]
```

SEE ALSO

NAME - ixSource

ixSource - recursive source

SYNOPSIS

ixSource *dirFileName*

DESCRIPTION

The **ixSource** command sources all the files in a particular folder and if there are sub-directories under the folder that are passed as an argument, it sources all the files under that sub-folder as well.

ARGUMENTS

dirFileName

(By value) Any number of files to be sourced or a folder name where all the files under that folder are going to be sourced.

RETURNS

None

EXAMPLES

```
ixSource test.tcl  
ixSource "c:/myTclProgs"
```

SEE ALSO

NAME - ixStartAtmOamTransmit

ixStartAtmOamTransmit - start ATM OAM transmit on a group of ports simultaneously

SYNOPSIS

ixStartAtmOamTransmit *portList*

DESCRIPTION

The **ixStartAtmOamTransmit** command sends a message to the IxServer to start ATM OAM message transmit on a group of ports simultaneously. The ports may span multiple chassis.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
if [ixStartAtmOamTransmit portList] {
    ixPuts $::ixErrorInfo
}
```

SEE ALSO

ixStartPortAtmOamTransmit, *ixStartAtmOamTransmit*,
ixStopPortAtmOamTransmit

NAME - ixStartCapture

ixStartCapture - start capture on a group of ports simultaneously

SYNOPSIS

ixStartCapture *portList*

DESCRIPTION

The **ixStartCapture** command sends a message to the IxServer to start capture on a group of ports simultaneously. The ports may span multiple chassis.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal
set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
set cardA 1
set portA 1
set cardB 1
set portB 2

# Examples of four ways to make a port list
set portList1 [list $chas,$cardA,$portA]
set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
```

```

set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

port setDefault
port set $chas $cardA $portA
port set $chas $cardB $portB

ixWritePortsToHardware one2oneArray
after 1000
if {[ixCheckLinkState one2oneArray] != 0} {
    ixPuts "Link is not up"
}

if {[ixStartCapture portList1] != 0} {
    ixPuts "Could not start capture on $portList1"
}
if {[ixStartCapture portList2] != 0} {
    ixPuts "Could not start capture on $portList2"
}
if {[ixStartCapture portList3] != 0} {
    ixPuts "Could not start capture on $portList3"
}
if {[ixStartCapture portList4] != 0} {
    ixPuts "Could not start capture on $portList4"
}
if {[ixStartCapture one2oneArray] != 0} {
    ixPuts "Could not start capture on $one2oneArray"
}

# Start transmit and wait a bit
ixStartTransmit one2oneArray
after 1000

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

ixStartPortCapture, *ixStopCapture*, *ixStopPortCapture*

NAME - ixStartCollisions

ixStartCollisions - start collision on a group of ports simultaneously

SYNOPSIS ixStartCollisions *portList*

DESCRIPTION The **ixStartCollisions** command sends a message to the IxServer to start collisions on a group of ports simultaneously. The ports may span multiple chassis. The ports must have been previously set-up for collisions by the **forceCollisions** command.

ARGUMENTS

portList (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1,1,2 1,1,3 1,1,4}

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA   1
set portA   1
set cardB   1
set portB   2
```

```

set portList [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
# Set up mapping
map new -type one2one
map config -type one2one
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

# Set up ports to 10Mbps and half duplex
port setDefault
port config -autonegotiate      false
port config -duplex            half
port config -speed             10
port set $chas $cardA $portA
port set $chas $cardB $portB

# Configure forced collisions
forcedCollisions setDefault
forcedCollisions config -enable 1
forcedCollisions config -consecutiveNonCollidingPackets 9
forcedCollisions set 1 $cardA $portA
forcedCollisions set 1 $cardB $portB

# Configure the streams to transmit at 50%
stream setDefault
stream config -percentPacketRate 50
stream config -rateMode          usePercentRate
stream set $chas $cardA $portA 1
stream set $chas $cardB $portB 1

# Write config to hardware, check the link state and clear
statistics
# Error checking omitted for brevity
ixWritePortsToHardware one2oneArray
after 1000
ixCheckLinkState one2oneArray
ixClearStats one2oneArray

ixPuts "Starting Transmit.."
ixStartStaggeredTransmit one2oneArray

ixPuts "Sleeping for 5 seconds"
after 5000

ixPuts "Awake. Now going to attempt to start collisions"

if {[ixStartCollisions ::one2oneArray] != 0} {
    ixPuts "Could not start collisions on $::one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $portList

```

```
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixStartPortCollisions, *ixStopCollisions*, *ixStopPortCollisions*

NAME - ixStartPacketGroups

ixStartPacketGroups - start calculating real-time latency on a group of ports simultaneously

SYNOPSIS

ixStartPacketGroups *portList*

DESCRIPTION

The **ixStartPacketGroups** command sends a message to the IxServer to start calculating real-time latency metrics on a group of ports simultaneously. The minimum, maximum and average latencies are calculated for each packet group ID (PGID). The ports may span multiple chassis. Ensure to clear timestamps on all send and receive ports before starting latency measurements.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1
```

```

set portList [list [list $chassis $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Set up port for loopback and packet group mode
port      setDefault
port      config -loopback      true
port      config -receiveMode   portPacketGroup
port      set $chassis $card $port

# Set up packet group configuration
packetGroup setDefault
packetGroup config -groupIdOffset      52
packetGroup config -latencyControl     cutThrough
packetGroup config -preambleSize       8
packetGroup config -signature          {08 71 18 05}
packetGroup config -signatureOffset    48
packetGroup setRx $chassis $card $port

# Configure fir (Frame Identification Record) for stream
stream      setDefault
stream      config -fir           true

# Set UDF 1 to count up the packet group
udf        setDefault
udf        config -enable         true
udf        config -continuousCount false
udf        config -countertype    c16
udf        config -initval        {00 00}
udf        config -offset          52
udf        config -repeat          10
udf        config -updown         uuuu
udf        set 1

# Write config to stream
stream      set      $chassis $card $port 1

# Set up packet group configuration
packetGroup setDefault
packetGroup config -groupId          1
packetGroup config -groupIdOffset    52
packetGroup config -insertSignature  true
packetGroup config -signature        {08 71 18 05}
packetGroup config -signatureOffset  48
packetGroup setTx $chassis $card $port 1

# Write config to hardware, error checking omitted for brevity
ixWritePortsToHardware portList
after 1000
ixCheckLinkState portList

# Start packet group operation
if {[ixStartPacketGroups portList] != 0} {
    ixPuts "Could not start packet groups on $portList"
}

```

```
}
```

```
# And then transmit
ixStartTransmit $portList
```

```
# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixStartPortPacketGroups, *ixStopPortPacketGroups*, *ixStopPacketGroups*

NAME - *ixStartPortAtmOamTransmit*

ixStartPortAtmOamTransmit - start ATM OAM transmit on an individual port

SYNOPSIS

ixStartPortAtmOamTransmit chassisID cardID portID

DESCRIPTION

The **ixStartPortAtmOamTransmit** command starts ATM OAM transmit on a single port.

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.

RETURNS

0	No error; the command was successfully delivered to the IxServer.
1	Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
if [ixStartPortAtmOamTransmit 1 2 1[ {  
    ixPuts $::ixErrorInfo  
}
```

SEE ALSO

ixStartAtmOamTransmit, *ixStopAtmOamTransmit*,
ixStopPortAtmOamTransmit

NAME - ixStartPortCapture

ixStartPortCapture - start capture on an individual port

SYNOPSIS

ixStartPortCapture *chassisID cardID portID*

DESCRIPTION

The **ixStartPortCapture** command starts capture on a single port.

ARGUMENTS

- | | |
|------------------|--|
| chassisID | (By value) The ID number of the chassis. |
| cardID | (By value) The ID number of the card. |
| portID | (By value) The ID number of the port. |

RETURNS

- | | |
|----------|--|
| 0 | No error; the command was successfully delivered to the IxServer. |
| 1 | Error; the command was delivered to the IxServer but it could not process the message. |

EXAMPLES

```
package require IxTclHal

set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
```

```
    ixPuts $::ixErrorInfo
    return 1
}

# Set loopback on port
port    setDefault
port    config -loopback true
port    set $chassis $card $port

# Set up stream to defaults
stream setDefault
stream set $chassis $card $port 1

# Write config to hardware and check link state
# Error checking omitted for brevity
ixWritePortsToHardware portList
after 1000
ixCheckLinkState portList

ixStartPortTransmit $chassis $card $port
if {[ixStartPortCapture $chassis $card $port] != 0} {
    ixPuts "Could not start port capture on $chassis:$card:$port"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixStartCapture, *ixStopCapture*, *ixStopPortCapture*

NAME - ixStartPortCollisions

ixStartPortCollisions - start collisions on an individual port

SYNOPSIS

ixStartPortCollisions *chassisID cardID portID*

DESCRIPTION

The **ixStartPortCollisions** command starts collisions on a single port.

ARGUMENTS

- chassisID** (By value) The ID number of the chassis.
- cardID** (By value) The ID number of the card.
- portID** (By value) The ID number of the port.

RETURNS

- 0** No error; the command was successfully delivered to the IxServer
- 1** Error; the command was delivered to the IxServer but it could not process the message

EXAMPLES

```

package require IxTclHal
set host      galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
set cardA   1
set portA   1
set cardB   1
set portB   2

set portList [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use

```

```

if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
# Set up mapping array
map new -type one2one
map config -type one2one
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

# Set up ports to 10Mbps and half duplex
port setDefault
port config -autonegotiate false
port config -duplex      half
port config -speed       10
port set $chas $cardA $portA
port set $chas $cardB $portB

# Configure forced collisions
forcedCollisions setDefault
forcedCollisions config -enable 1
forcedCollisions config -consecutiveNonCollidingPackets 9
forcedCollisions set $chas $cardA $portA
forcedCollisions set $chas $cardB $portB
# Configure the streams to transmit at 50%
stream setDefault
stream config -percentPacketRate 50
stream config -rateMode          usePercentRate
stream set $chas $cardA $portA 1
stream set $chas $cardB $portB 1

# Write config to hardware, check the link state and clear
statistics
# Error checking omitted for brevity
ixWritePortsToHardware one2oneArray
after 1000
ixCheckLinkState one2oneArray
ixClearStats one2oneArray

ixPuts "Starting Transmit.."
ixStartStaggeredTransmit one2oneArray

ixPuts "Sleeping for 5 seconds"
after 5000
ixPuts "Awake. Now going to attempt to start collisions"

if {[ixStartPortCollisions $chas $cardA $portA] != 0} {
    ixPuts "Could not start collisions on $chas:$card:$port"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixStartCollisions*, *ixStopCollisions*, *ixStopPortCollisions*

NAME - ixStartPortPacketGroups

ixStartPortPacketGroups - start packet group operations on an individual port

SYNOPSIS

ixStartPortPacketGroups *chassisID cardID portID*

DESCRIPTION

The **ixStartPortPacketGroups** command sends a message to the IxServer to start calculating real-time latency metrics on a single port. The minimum, maximum and average latencies are calculated for each packet group ID (PGID).

ARGUMENTS

- | | |
|------------------|--|
| chassisID | (By value) The ID number of the chassis. |
| cardID | (By value) The ID number of the card. |
| portID | (By value) The ID number of the port. |

RETURNS

- | | |
|----------|--|
| 0 | No error; the command was successfully delivered to the IxServer. |
| 1 | Error; the command was delivered to the IxServer but it could not process the message. |

EXAMPLES

```

package require IxTclHal
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1

set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
}

```

```

        return 1
    }
    # Take ownership of the ports we'll use
    if [ixTakeOwnership $portList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Set up port for loopback and packet group mode
    port      setDefault
    port      config -loopback      true
    port      config -receiveMode   portPacketGroup
    port      set $chassis $card $port

    # Set up packet group configuration
    packetGroup setDefault
    packetGroup config -groupIdOffset 52
    packetGroup config -latencyControl cutThrough
    packetGroup config -preambleSize 8
    packetGroup config -signature {08 71 18 05}
    packetGroup config -signatureOffset 48
    packetGroup setRx $chassis $card $port

    # Configure fir (Frame Identification Record) for stream
    stream      setDefault
    stream      config -fir          true

    # Set UDF 1 to count up the packet group
    udf        setDefault
    udf        config -enable        true
    udf        config -continuousCount false
    udf        config -countertype   c16
    udf        config -initval      {00 00}
    udf        config -offset       52
    udf        config -repeat       10
    udf        config -updown       uuuu
    udf        set           1

    # Write config to stream
    stream      set      $chassis $card $port 1

    # Set up packet group configuration
    packetGroup setDefault
    packetGroup config -groupId      1
    packetGroup config -groupIdOffset 52
    packetGroup config -insertSignature true
    packetGroup config -signature    {08 71 18 05}
    packetGroup config -signatureOffset 48
    packetGroup setTx $chassis $card $port 1

    # Write config to hardware, error checking omitted for brevity
    ixWritePortsToHardware $portList
    after 1000
    ixCheckLinkState $portList

    # Start packet group operation
    if {[ixStartPortPacketGroups $chassis $card $port] != 0} {
        ixPuts "Could not start packet groups on $chassis:$card:$port"
    }

    # And then transmit
    ixStartTransmit $portList

```

```
# Let go of the ports that we reserved  
ixClearOwnership $portList  
# Disconnect from the chassis we're using  
ixDisconnectFromChassis $host  
# If we're running on UNIX, disconnect from the TCL Server  
if [isUNIX] {  
    ixDisconnectTclServer $host  
}
```

SEE ALSO

ixStartPacketGroups, *ixStopPacketGroups*, *ixStopPortPacketGroups*

NAME - ixStartPortTransmit

ixStartPortTransmit - start transmission on an individual port

SYNOPSIS ixStartPortTransmit *chassisID cardID portID*

DESCRIPTION The **ixStartPortTransmit** command starts transmission on a single port.

ARGUMENTS

chassisID (By value) The ID number of the chassis.
cardID (By value) The ID number of the card.
portID (By value) The ID number of the port.

RETURNS

0 No error; the command was successfully delivered to the IxServer.
1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```
}

# Set loopback on port
port    setDefault
port    config -loopback true
port    set $chas $card $port

# Set up stream to defaults
stream setDefault
stream set $chas $card $port 1

# Write config to hardware and check link state
# Error checking omitted for brevity
ixWritePortsToHardware portList
after 1000
ixCheckLinkState portList

if {[ixStartPortTransmit $chas $card $port] != 0} {
    ixPuts "Could not start port transmit on $chas:$card:$port"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixStartTransmit, *ixStopTransmit*, *ixStopTransmit*

NAME - ixStartStaggeredTransmit

ixStartStaggeredTransmit - start transmission on a group of ports in sequence

SYNOPSIS	ixStartStaggeredTransmit <i>portList</i>
-----------------	--

DESCRIPTION	The ixStartStaggeredTransmit command sends a message to the IxServer to start transmission on a group of ports in sequence. The ports may span multiple chassis.
--------------------	---

ARGUMENTS

portList	(By reference) The list of ports in one of the following formats:
-----------------	---

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0	No error; the command was successfully delivered to the IxServer.
----------	---

1	Error; the command was delivered to the IxServer but it could not process the message.
----------	--

EXAMPLES

```
package require IxTclHal
set host      galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA   1
set portA   1
set cardB   1
set portB   2

# Examples of four ways to make a port list
```

```

set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

port setDefault
port set $chas $cardA $portA
port set $chas $cardB $portB

ixWritePortsToHardware one2oneArray
if {[ixCheckLinkState one2oneArray] != 0} {
    ixPuts "Link is not up"
}

if {[ixStartStaggeredTransmit portList2] != 0} {
    ixPuts "Could not start StaggeredTransmit on $portList2"
}
if {[ixStartStaggeredTransmit portList3] != 0} {
    ixPuts "Could not start StaggeredTransmit on $portList3"
}
if {[ixStartStaggeredTransmit portList4] != 0} {
    ixPuts "Could not start StaggeredTransmit on $portList4"
}
if {[ixStartStaggeredTransmit one2oneArray] != 0} {
    ixPuts "Could not start StaggeredTransmit on $one2oneArray"
}

after 1000

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

ixStartTransmit, *ixStopTransmit*, *ixStartPortTransmit*, *ixStopPortTransmit*

NAME - ixStartTransmit

ixStartTransmit - start transmission on a group of ports simultaneously

SYNOPSIS	ixStartTransmit <i>portList</i>
-----------------	---------------------------------

DESCRIPTION	The ixStartTransmit command sends a message to the IxServer to start transmission on a group of ports simultaneously. The ports may span multiple chassis.
--------------------	---

ARGUMENTS

portList	(By reference) The list of ports in one of the following formats:
-----------------	---

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0	No error; the command was successfully delivered to the IxServer.
----------	---

1	Error; the command was delivered to the IxServer but it could not process the message.
----------	--

EXAMPLES

```
package require IxTclHal
set host      galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA   1
set portA   1
set cardB   1
set portB   2

# Examples of four ways to make a port list
```

```

set portList1 [list $chas,$cardA,$portA]
set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chas $cardA $portA      $chas $cardB $portB
map add      $chas $cardB $portB      $chas $cardA $portA

port setDefault
port set $chas $cardA $portA
port set $chas $cardB $portB

ixWritePortsToHardware one2oneArray
if {[ixCheckLinkState one2oneArray] != 0} {
    ixPuts "Link is not up\n"
    exit
}

if {[ixStartTransmit portList1] != 0} {
    ixPuts "Could not start Transmit on $portList1"
}
if {[ixStartTransmit portList2] != 0} {
    ixPuts "Could not start Transmit on $portList2"
}
if {[ixStartTransmit portList3] != 0} {
    ixPuts "Could not start Transmit on $portList3"
}
if {[ixStartTransmit portList4] != 0} {
    ixPuts "Could not start Transmit on $portList4"
}
if {[ixStartTransmit one2oneArray] != 0} {
    ixPuts "Could not start Transmit on $one2oneArray"
}

after 1000

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

ixStopTransmit, *ixStartPortTransmit*, *ixStopPortTransmit*

NAME - **ixStopAtmOamTransmit**

ixStopAtmOamTransmit - stop ATM OAM transmit on a group of ports simultaneously

SYNOPSIS *ixStopAtmOamTransmit portList*

DESCRIPTION The **ixStopAtmOamTransmit** command sends a message to the IxServer to stop ATM OAM message transmit on a group of ports simultaneously. The ports may span multiple chassis.

ARGUMENTS

portList (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
if [ixStopAtmOamTransmit portList] {  
    ixPuts $::ixErrorInfo  
}
```

SEE ALSO *ixStartPortAtmOamTransmit*, *ixStopAtmOamTransmit*,
 ixStopPortAtmOamTransmit

NAME - ixStopCapture

ixStopCapture - stop capture on a group of ports simultaneously

SYNOPSIS

ixStopCapture *portList*

DESCRIPTION

The **ixStopCapture** command sends a message to the IxServer to stop capture on a group of ports simultaneously. The ports may span multiple chassis.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal
set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
set cardA 1
set portA 1
set cardB 1
set portB 2

# Examples of four ways to make a port list
set portList1 [list $chas,$cardA,$portA]
set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
```

```

set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]

# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}
map new -type one2one
map config -type one2one
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

port setDefault
port set $chas $cardA $portA
port set $chas $cardB $portB

ixWritePortsToHardware one2oneArray
after 1000
if {[ixCheckLinkState one2oneArray] != 0} {
    ixPuts "Link is not up"
}

if {[ixStopCapture portList1] != 0} {
    ixPuts "Could not Stop capture on $portList1"
}
if {[ixStopCapture portList2] != 0} {
    ixPuts "Could not Stop capture on $portList2"
}
if {[ixStopCapture portList3] != 0} {
    ixPuts "Could not Stop capture on $portList3"
}
if {[ixStopCapture portList4] != 0} {
    ixPuts "Could not Stop capture on $portList4"
}
if {[ixStopCapture one2oneArray] != 0} {
    ixPuts "Could not Stop capture on $one2oneArray"
}
# Stop transmit and wait a bit
ixStopTransmit one2oneArray
after 1000

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixStartCapture, ixStartPortCapture, ixStopPortCapture*

NAME - ixStopCollisions

ixStopCollisions - stop collisions on a group of ports simultaneously

SYNOPSIS

ixStopCollisions *portList*

DESCRIPTION

The **ixStopCollisions** command sends a message to the IxServer to stop collisions on a group of ports simultaneously. The ports may span multiple chassis.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal
set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA    1
set portA   1
set cardB    1
set portB   2
```

```

set portList [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}
# Set up mapping
map new -type one2one
map config -type one2one
map add     $chas $cardA $portA   $chas $cardB $portB
map add     $chas $cardB $portB   $chas $cardA $portA

# Set up ports to 10Mbps and half duplex
port setDefault
port config -autonegotiate      false
port config -duplex            half
port config -speed             10
port set $chas $cardA $portA
port set $chas $cardB $portB

# Configure forced collisions
forcedCollisions setDefault
forcedCollisions config -enable 1
forcedCollisions config -consecutiveNonCollidingPackets 9
forcedCollisions set 1 $cardA $portA
forcedCollisions set 1 $cardB $portB

# Configure the streams to transmit at 50%
stream setDefault
stream config -percentPacketRate 50
stream config -rateMode          usePercentRate
stream set $chas $cardA $portA 1
stream set $chas $cardB $portB 1

# Write config to hardware, check the link state and clear
statistics
# Error checking omitted for brevity
ixWritePortsToHardware one2oneArray
after 1000
ixCheckLinkState one2oneArray
ixClearStats one2oneArray

ixStartStaggeredTransmit one2oneArray
after 1000
ixStartCollisions one2oneArray

after 1000
if {[ixStopCollisions one2oneArray] != 0} {
    ixPuts "Could not stop collisions on $one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {

```

```
    ixDisconnectTclServer $host  
}
```

SEE ALSO

ixStartCollisions, *ixStartPortCollisions*, *ixStopPortCollisions*

NAME - ixStopPacketGroups

ixStopPacketGroups - stop calculating real-time latency on a group of ports simultaneously

SYNOPSIS ixStopPacketGroups *portList*

DESCRIPTION The **ixStopPacketGroups** command sends a message to the IxServer to stop calculating real-time latency metrics on a group of ports simultaneously. The ports may span multiple chassis.

ARGUMENTS

portList (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1

set portList [list [list $chas $cardA $portA]]
# Login before taking ownership
```

```

if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Set up port for loopback and packet group mode
port      setDefault
port      config -loopback      true
port      config -receiveMode   portPacketGroup
port      set $chassis $card $port

# Set up packet group configuration
packetGroup setDefault
packetGroup config -groupIdOffset 52
packetGroup config -latencyControl cutThrough
packetGroup config -preambleSize 8
packetGroup config -signature {08 71 18 05}
packetGroup config -signatureOffset 48
packetGroup setRx $chassis $card $port

# Configure fir (Frame Identification Record) for stream
stream      setDefault
stream      config -fir          true

# Set UDF 1 to count up the packet group
udf        setDefault
udf        config -enable        true
udf        config -continuousCount false
udf        config -countertype   c16
udf        config -initval      {00 00}
udf        config -offset        52
udf        config -repeat        10
udf        config -updown       uuuu
udf        set 1

# Write config to stream
stream      set $chassis $card $port 1

# Set up packet group configuration
packetGroup setDefault
packetGroup config -groupId      1
packetGroup config -groupIdOffset 52
packetGroup config -insertSignature true
packetGroup config -signature     {08 71 18 05}
packetGroup config -signatureOffset 48
packetGroup setTx $chassis $card $port 1

# Write config to hardware, error checking omitted for brevity
ixWritePortsToHardware $portList
after 1000
ixCheckLinkState $portList

# Start packet group operation
ixStartPacketGroups $portList

# And then transmit
ixStartTransmit $portList
after 10000

```

```
if {[ixStopPacketGroups portList] != 0} {
    ixPuts "Can't stop packet group operation on $portList"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixStartPacketGroups, *ixStartPortPacketGroups*, *ixStopPortPacketGroups*

NAME - ixStopPortAtmOamTransmit

ixStopPortAtmOamTransmit - stop ATM OAM transmit on an individual port

SYNOPSIS

ixStopPortAtmOamTransmit *chassisID cardID portID*

DESCRIPTION

The **ixStopPortAtmOamTransmit** command stops ATM OAM transmit on a single port.

ARGUMENTS

- chassisID** (By value) The ID number of the chassis.
cardID (By value) The ID number of the card.
portID (By value) The ID number of the port.

RETURNS

- 0** No error; the command was successfully delivered to the IxServer.
1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
if [ixStopPortAtmOamTransmit 1 2 1] {  
    ixPuts $::ixErrorInfo  
}
```

SEE ALSO

ixStartAtmOamTransmit, *ixStopAtmOamTransmit*,
ixStartPortAtmOamTransmit

NAME - ixStopPortCapture

ixStopPortCapture - stop capture on an individual port

SYNOPSIS ixStopPortCapture *chassisID cardID portID [groupId] [create] [destroy]*

DESCRIPTION The **ixStopPortCapture** command stops capture on a single port.

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.
groupId	(By value) The group number to be used in the join message. If omitted, the default value of 101064 is used.
create	(By value) Create a new port group (<i>create</i>) or not (<i>nocreate</i>). (<i>default = create</i>)
destroy	(By value) Clean up a created port group when command completes (<i>destroy</i>) or not (<i>nodestroy</i>). (<i>default = destroy</i>)

RETURNS

0	No error; the command was successfully delivered to the IxServer.
1	Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal
set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA 1
```

```

set portA    1
set cardB   1
set portB    2
set portList [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

port setDefault
port set $chas $cardA $portA
port set $chas $cardB $portB

ixWritePortsToHardware one2oneArray
after 1000
if {[ixCheckLinkState one2oneArray] != 0} {
    ixPuts "Link is not up"
}

if {[ixStartCapture one2oneArray] != 0} {
    ixPuts "Could not start capture on $one2oneArray"
}

# Start transmit and wait a bit
ixStartTransmit one2oneArray
after 1000

if {[ixStopPortCapture $chas $cardA $portA] != 0} {
    ixPuts "Could not stop capture on $chas:$cardA:$portA"
}
if {[ixStopPortCapture $chas $cardB $portB] != 0} {
    ixPuts "Could not stop capture on $chas:$cardB:$portB"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO

ixStartCapture, *ixStopCapture*, *ixStartPortCapture*

NAME - ixStopPortCollisions

ixStopPortCollisions - stop collisions on an individual port

SYNOPSIS ixStopPortCollisions *chassisID cardID portID*

DESCRIPTION The **ixStopPortCollisions** command stops collisions on a single port.

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.

RETURNS

0	No error; the command was successfully delivered to the IxServer.
1	Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```

package require IxTclHal
set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]
set cardA 1
set portA 1
set cardB 1
set portB 2
set portList [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {

```

```

        ixPuts $::ixErrorInfo
        return 1
    }
    # Set up mapping
    map new -type one2one
    map config -type one2one
    map add      $chas $cardA $portA  $chas $cardB $portB
    map add      $chas $cardB $portB  $chas $cardA $portA

    # Set up ports to 10Mbps and half duplex
    port setDefault
    port config -autonegotiate      false
    port config -duplex            half
    port config -speed             10
    port set $chas $cardA $portA
    port set $chas $cardB $portB

    # Configure forced collisions
    forcedCollisions setDefault
    forcedCollisions config -enable 1
    forcedCollisions config -consecutiveNonCollidingPackets 9
    forcedCollisions set $chas $cardA $portA
    forcedCollisions set $chas $cardB $portB

    # Configure the streams to transmit at 50%
    stream setDefault
    stream config -percentPacketRate 50
    stream config -rateMode          usePercentRate
    stream set $chas $cardA $portA 1
    stream set $chas $cardB $portB 1

    # Write config to hardware, check the link state and clear
    # statistics
    # Error checking omitted for brevity
    ixWritePortsToHardware one2oneArray
    after 1000
    ixCheckLinkState one2oneArray
    ixClearStats one2oneArray

    ixStartStaggeredTransmit one2oneArray
    after 1000
    ixStartCollisions ::one2oneArray

    after 1000
    if {[ixStopPortCollisions $chas $cardA $portA] != 0} {
        ixPuts "Could not stop collisions on $chas:$cardA:$portA"
    }
    if {[ixStopPortCollisions $chas $cardB $portB] != 0} {
        ixPuts "Could not stop collisions on $chas:$cardB:$portB"
    }
    # Let go of the ports that we reserved
    ixClearOwnership $portList
    # Disconnect from the chassis we're using
    ixDisconnectFromChassis $host
    # If we're running on UNIX, disconnect from the TCL Server
    if [isUNIX] {
        ixDisconnectTclServer $host
    }
}

```

SEE ALSO*ixStartCollisions*, *ixStopCollisions*, *ixStartPortCollisions*

NAME - ixStopPortPacketGroups

ixStopPortPacketGroups - stop packet group operations on an individual port

SYNOPSIS	ixStopPortPacketGroups <i>chassisID cardID portID</i>
-----------------	---

DESCRIPTION	The ixStopPortPacketGroups command sends a message to the IxServer to stop calculating real-time latency metrics on a single port. The minimum, maximum and average latencies are calculated for each packet group ID (PGID).
--------------------	--

ARGUMENTS

chassisID	(By value) The ID number of the chassis.
cardID	(By value) The ID number of the card.
portID	(By value) The ID number of the port.

RETURNS

0	No error; the command was successfully delivered to the IxServer.
1	Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```

package require IxTclHal
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1

set portList [list [list $chas $cardA $portA]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
}

```

```

        return 1
    }
    # Take ownership of the ports we'll use
    if [ixTakeOwnership $portList] {
        ixPuts $::ixErrorInfo
        return 1
    }

    # Set up port for loopback and packet group mode
    port      setDefault
    port      config -loopback      true
    port      config -receiveMode   portPacketGroup
    port      set $chash $card $port

    # Set up packet group configuration
    packetGroup setDefault
    packetGroup config -groupIdOffset 52
    packetGroup config -latencyControl cutThrough
    packetGroup config -preambleSize 8
    packetGroup config -signature {08 71 18 05}
    packetGroup config -signatureOffset 48
    packetGroup setRx $chash $card $port

    # Configure fir (Frame Identification Record) for stream
    stream      setDefault
    stream      config -fir          true

    # Set UDF 1 to count up the packet group
    udf        setDefault
    udf        config -enable        true
    udf        config -continuousCount false
    udf        config -countertype   c16
    udf        config -initval      {00 00}
    udf        config -offset       52
    udf        config -repeat       10
    udf        config -updown      uuuu
    udf        set      1

    # Write config to stream
    stream      set      $chash $card $port 1

    # Set up packet group configuration
    packetGroup setDefault
    packetGroup config -groupId      1
    packetGroup config -groupIdOffset 52
    packetGroup config -insertSignature true
    packetGroup config -signature     {08 71 18 05}
    packetGroup config -signatureOffset 48
    packetGroup setTx $chash $card $port 1

    # Write config to hardware, error checking omitted for brevity
    ixWritePortsToHardware portList
    after 1000
    ixCheckLinkState portList

    # Start packet group operation
    ixStartPortPacketGroups $chash $cardA $portA
    ixStartPortPacketGroups $chash $cardB $portB

    # And then transmit
    ixStartTransmit portList
    after 10000

```

```
if {[ixStopPortPacketGroups $chas $cardA $portA] != 0} {
    ixPuts "Can't stop packet group operation on
$chas:$cardA:$portA"
}
if {[ixStopPortPacketGroups $chas $cardB $portB] != 0} {
    ixPuts "Can't stop packet group operation on
$chas:$cardB:$portB"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixStartPacketGroups, *ixStopPacketGroups*, *ixStartPortPacketGroups*

NAME - ixStopPortTransmit

ixStopPortTransmit - stop transmission on an individual port

SYNOPSIS `ixStopPortTransmit chassisID cardID portID`

DESCRIPTION The **ixStopPortTransmit** command stops transmission on a single port.

ARGUMENTS

- chassisID** (By value) The ID number of the chassis.
- cardID** (By value) The ID number of the card.
- portID** (By value) The ID number of the port.

RETURNS

- 0** No error; the command was successfully delivered to the IxServer.
- 1** Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```

package require IxTclHal
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set card 1
set port 1
set portList [list [list $chas $card $port]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

```

```
# Set loopback on port
port setDefault
port config -loopback true
port set $chassis $card $port
# Set up stream to defaults
stream setDefault
stream set $chassis $card $port 1

# Write config to hardware and check link state
# Error checking omitted for brevity
ixWritePortsToHardware portList
after 1000
ixCheckLinkState portList

if {[ixStartPortTransmit $chassis $card $port] != 0} {
    ixPuts "Could not start port transmit on $chassis:$card:$port"
}
after 1000
if {[ixStopPortTransmit $chassis $card $port] != 0} {
    ixPuts "Could not stop port transmit on $chassis:$card:$port"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO

ixStartTransmit, *ixStopTransmit*, *ixStopPortTransmit*

NAME - ixStopTransmit

ixStopTransmit - stop transmission on a group of ports simultaneously

SYNOPSIS `ixStopTransmit portList`

DESCRIPTION The **ixStopTransmit** command stops transmission on a single port.

ARGUMENTS

portList (By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
`set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}}` -or-
`set pl {1,1,1 1,1,2 1,1,3 1,1,4}`

RETURNS

0 No error; the command was successfully delivered to the IxServer.

1 Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA   1
set portA   1
set cardB   1
set portB   2

# Examples of four ways to make a port list
set portList1 [list $chas,$cardA,$portA]
set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]
# Login before taking ownership
```

```

if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

port setDefault
port set $chas $cardA $portA
port set $chas $cardB $portB

ixWritePortsToHardware one2oneArray
after 1000
if {[ixCheckLinkState one2oneArray] != 0} {
    ixPuts "Link is not up"
}

ixStartTransmit portList1
after 5000
if {[ixStopTransmit portList1] != 0} {
    ixPuts "Could not stop Transmit on $portList1"
}
ixStartTransmit portList2
after 5000
if {[ixStopTransmit portList2] != 0} {
    ixPuts "Could not stop Transmit on $portList2"
}
ixStartTransmit portList3
after 5000
if {[ixStopTransmit portList3] != 0} {
    ixPuts "Could not stop Transmit on $portList3"
}
ixStartTransmit portList4
after 5000
if {[ixStopTransmit portList4] != 0} {
    ixPuts "Could not stop Transmit on $portList4"
}
ixStartTransmit ::one2oneArray
after 5000
if {[ixStopTransmit one2oneArray] != 0} {
    ixPuts "Could not stop Transmit on $one2oneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixStopTransmit, ixStartPortTransmit, ixStopPortTransmit*

NAME - ixTakeOwnership

ixTakeOwnership - takes ownership of all the ports in the list

SYNOPSIS

ixTakeOwnership *portList* [*takeType*]

DESCRIPTION

The **ixTakeOwnership** command takes ownership of all the ports in the list.

When a list of ports is supplied to ixTakeOwnership and one of the ports does not exist, the command takes ownership where it can, and prints a message line for the port that it cannot take ownership, and returns a 0. The port for which ownership cannot be taken is removed from the list, and the process continues.

This message is posted:

Port [getPortId \$c \$1 \$p] is not available, removing port from the list.

A value of 1 is returned when ixTakeOwnership is given just one, non-existent port as a parameter.

ARGUMENTS

portList

(By value) The list of ports in one of the following formats:

One of the following literal strings, or a reference to a variable with the \$ (for example, \$pl after set pl ...)
 {{1 1 1}}
 {{1 1 1} {1 1 2} {1 1 3} {1 1 4}}
 {{1 1 *} {1 2 1} {1 2 2}}

takeType

(By value) (*Optional*) Valid values:

force: take regardless of whether the port is owned by someone else
notForce: do not force ownership

RETURNS

0

No error; the command was successfully delivered to the IxServer. Ownership of at least one port (in the list) was successfully acquired.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal
set host galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}
```

```
}
```

```
# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set portsToOwn {{$chas 1 *} {$chas 2 1} {$chas 2 2}}
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portsToOwn force] {
    ixPuts $::ixErrorInfo
    return 1
}
```

SEE ALSO

ixClearOwnership, *ixPortClearOwnership*, *ixPortTakeOwnership*

NAME - ixTransmitArpRequest

ixTransmitArpRequest - transmit ARP requests on a group of ports simultaneously

SYNOPSIS

ixTransmitArpRequest *portList*

DESCRIPTION

The **ixTransmitArpRequest** command sends a message to the IxServer to start transmission of ARP requests on a group of ports simultaneously using the protocol server. The ports may span multiple chassis.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
 Or a reference to a list. For example, *pl* after
 set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
 set pl {1,1,1 1,1,2 1,1,3 1,1,4}

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host    galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA  1
set portA  1
set cardB  1
set portB  2
```

```

# Four different port list formats
set portList1 [list $chas,$cardA,$portA]
set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type onezone
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

# Need to set up IP for ARP
ip setDefault
ip set 1 1 1
ip set 1 1 2

# Try each of the formats
if {[ixTransmitArpRequest portList1] != 0} {
    ixPuts "Could not transmit ARP request for $portList1\n"
}
if {[ixTransmitArpRequest portList2] != 0} {
    ixPuts "Could not transmit ARP request for $portList2\n"
}
if {[ixTransmitArpRequest portList3] != 0} {
    ixPuts "Could not transmit ARP request for $portList3\n"
}
if {[ixTransmitArpRequest portList4] != 0} {
    ixPuts "Could not transmit ARP request for $portList4\n"
}
if {[ixTransmitArpRequest one2oneArray] != 0} {
    ixPuts "Could not transmit ARP request for $one2oneArray\n"
}

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixTransmitPortArpRequest*

NAME - ixTransmitPortArpRequest**ixTransmitPortArpRequest** - transmit ARP requests on an individual port**SYNOPSIS**ixTransmitPortArpRequest *chassisID cardID portID***DESCRIPTION**The **ixTransmitPortArpRequest** command sends a message to the IxServer to start transmission of ARP requests on a single port using the protocol server.**ARGUMENTS**

- chassisID** (By value) The ID number of the chassis
- cardID** (By value) The ID number of the card
- portID** (By value) The ID number of the port

RETURNS

- 0** No error; the command was successfully delivered to the IxServer.
- 1** Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA   1
set portA   1
set cardB   1
set portB   2
set portList [list [list $chas $cardA $portA] [list $chas $cardB
$portB]]
# Login before taking ownership
if [ixLogin $username] {
```

```
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList] {
    ixPuts $::ixErrorInfo
    return 1
}

# Need to set up IP for ARP
ip setDefault
ip set $chas $cardA $portA
ip set $chas $cardB $portB

if {[ixTransmitPortArpRequest $chas $cardA $portA] != 0} {
    ixPuts "Could not transmit ARP request for
$chas:$cardA:$portA"
}
if {[ixTransmitPortArpRequest $chas $cardB $portB] != 0} {
    ixPuts "Could not transmit ARP request for
$chas:$cardB:$portB"
}

# Let go of the ports that we reserved
ixClearOwnership $portList
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}
```

SEE ALSO*ixTransmitArpRequest*

NAME - ixUtils

ixUtils - determine whether optional software components are installed

SYNOPSIS

ixUtils *sub-command*

DESCRIPTION

The **ixUtils** sub-commands allow for the determination whether optional software has been installed.

COMMANDS

The **ixUtils** command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.

Each of the ixUtils sub-commands are available as separate commands. The following table indicates the equivalence.

ixUtils sub-command	High-Level API command
ixUtils calculateMaxRate	<i>calculateMaxRate</i>
ixUtils calculatePercentMaxRate	<i>calculatePercentMaxRate</i>
ixUtils getErrorString	<i>getErrorResponse</i>

EXAMPLES

See examples under *calculateMaxRate*, *calculatePercentMaxRate*.

SEE ALSO

calculateMaxRate, *calculatePercentMaxRate*, *getErrorResponse*

NAME - ixWriteConfigToHardware

ixWriteConfigToHardware - writes streams, filters, protocol configuration on ports in hardware

SYNOPSIS

```
ixWriteConfigToHardware portList [-verbose | -noVerbose]
[-writeProtocolServer | -noProtocolServer]
```

DESCRIPTION

The **ixWriteConfigToHardware** command commits the configuration of streams, filters, and protocol information on a group of ports to hardware. This command is useful when a large number of ports are involved.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

-verbose | -noVerbose

(Optional). Either *noVerbose* (*default*) or *verbose*, which appends a status message to the log file.

-writeProtocolServer | -noProtocolServer

(Optional) *-writeProtocolServer* stops the protocol server and writes all associated objects (*default*). *-noProtocolServer* has no effect on the protocol server and does not update any protocol server objects.

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

```
package require IxTclHal

set host      galaxy
set username user
# Check if we're running on UNIX - connect to the TCL Server
# which must be running on the chassis
if [isUNIX] {
    if [ixConnectToTclServer $host] {
        ixPuts "Could not connect to $host"
        return 1
    }
}

# Now connect to the chassis
if [ixConnectToChassis $host] {
    ixPuts $::ixErrorInfo
    return 1
}
```

```

# Get the chassis ID to use in port lists
set chas [ixGetChassisID $host]

set cardA 1
set portA 1
set cardB 1
set portB 2

# Examples of four ways to make a port list
set portList1 [list $chas,$cardA,$portA]
set portList2 [list $chas,$cardA,$portA $chas,$cardB,$portB]
set portList3 [list [list $chas $cardA $portA] [list $chas $cardB $portB]]
set portList4 [list [list $chas,$cardA,$portA] [list
$chas,$cardB,$portB]]
# Login before taking ownership
if [ixLogin $username] {
    ixPuts $::ixErrorInfo
    return 1
}
# Take ownership of the ports we'll use
if [ixTakeOwnership $portList4] {
    ixPuts $::ixErrorInfo
    return 1
}

map new -type one2one
map config -type one2one
map add      $chas $cardA $portA   $chas $cardB $portB
map add      $chas $cardB $portB   $chas $cardA $portA

port setDefault
port set $chas $cardA $portA
port set $chas $cardB $portB

if {[ixWriteConfigToHardware portList1] != 0} {
    ixPuts "Could not write config to $portList1"
}
if {[ixWriteConfigToHardware portList2] != 0} {
    ixPuts "Could not write config to $portList2"
}
if {[ixWriteConfigToHardware portList3] != 0} {
    ixPuts "Could not write config to $portList3"
}
if {[ixWriteConfigToHardware portList4] != 0} {
    ixPuts "Could not write config to $portList4"
}
if {[ixWriteConfigToHardware oneZoneArray] != 0} {
    ixPuts "Could not write config to oneZoneArray"
}

# Let go of the ports that we reserved
ixClearOwnership $portList4
# Disconnect from the chassis we're using
ixDisconnectFromChassis $host
# If we're running on UNIX, disconnect from the TCL Server
if [isUNIX] {
    ixDisconnectTclServer $host
}

```

SEE ALSO*ixWritePortsToHardware*

NAME - *ixWritePortsToHardware*

ixWritePortsToHardware - writes port properties in hardware

SYNOPSIS

*ixWritePortsToHardware portList [-verbose | -noVerbose]
[-writeProtocolServer | -noProtocolServer]*

DESCRIPTION

The **ixWritePortsToHardware** command commits the configuration such as Mii properties on 10/100 interface (such as speed, duplex modes, auto negotiation), port properties on Gigabit interfaces, and PPP parameters on Packet over Sonet interfaces on a group of ports to hardware. It also performs all of the functions of *ixWriteConfigToHardware*. This command is useful when a large number of ports are involved. Note, this command may result in a loss of link, depending on the changes that have been made.

ARGUMENTS

portList

(By reference) The list of ports in one of the following formats:

one2oneArray, one2manyArray, many2oneArray, many2manyArray
Or a reference to a list. For example, *pl* after
set pl {{1 1 1} {1 1 2} {1 1 3} {1 1 4}} -or-
set pl {1,1,1 1,1,2 1,1,3 1,1,4}

-verbose | -noVerbose

(Optional). Either *noVerbose* (*default*) or *verbose*, which appends a status message to the log file.

-writeProtocolServer | -noProtocolServer

(Optional) *-writeProtocolServer* stops the protocol server and writes all associated objects (*default*). *-noProtocolServer* has no effect on the protocol server and does not update any protocol server objects.

RETURNS

0

No error; the command was successfully delivered to the IxServer.

1

Error; the command was delivered to the IxServer but it could not process the message.

EXAMPLES

See the example under *ixStartTransmit*.

SEE ALSO

ixWriteConfigToHardware

NAME - map

map - configure traffic map.

SYNOPSIS	map <i>sub-command options</i>
DESCRIPTION	<p>The map command is used to set the direction of traffic flow between ports on same or different cards on same or different chassis. There are four types of mappings available - one2one, one2many, many2one and many2many.</p> <p>The <i>one2one</i> mapping sets up one transmit and one receive port for traffic flow. The transmit/receive port pair that has been configured once cannot be used in a different port pair. That is, each port pair is mutually exclusive. The <i>one2many</i> mapping sets up one transmit port and multiple receive ports. Each group of transmit and its multiple receive ports is mutually exclusive with other groups. The <i>many2one</i> mapping sets up multiple transmit ports and one receive port. Each group of multiple transmit ports and its receive port is mutually exclusive with other groups. The <i>many2many</i> mapping sets up multiple transmit ports and multiple receive ports. Any port may transmit and receive to any other port in any group of ports.</p>
<hr/>	
STANDARD OPTIONS	
<p>type maptype <i>maptype</i> may be one of:</p> <ul style="list-style-type: none"> <i>one2one</i> <i>one2many</i> <i>many2one</i> <i>many2many</i> 	
<hr/>	
COMMAND	<p>The map command is invoked with the following sub-commands. If no sub-command is specified, returns a list of all sub-commands available.</p>
<p>map add <i>txChassis txLm txPort rxChassis rxLm rxPort</i></p> <p>Creates a map from Tx ports <i>txPort</i> on card <i>txLm</i>, chassis <i>txChassis</i> to Rx port <i>rxPort</i> on card <i>rxLm</i>, chassis <i>rxChassis</i>.</p>	
<p>map cget <i>option</i></p> <p>Returns the current value of the configuration option given by <i>option</i>. <i>Option</i> may have any of the values accepted by the map command.</p>	
<p>map config <i>option value</i></p> <p>Modify the configuration options of the map. If no <i>option</i> is specified, returns a list describing all of the available options (see STANDARD OPTIONS) for map.</p>	
<p>map del <i>txChassis txLm txPort rxChassis rxLm rxPort</i></p> <p>Deletes a map from Tx ports <i>txPort</i> on card <i>txLm</i>, chassis <i>txChassis</i> to Rx port <i>rxPort</i> on card <i>rxLm</i>, chassis <i>rxChassis</i>.</p>	

map new -type type

Clears the current map of type *one2one*, *one2many*, *many2one*, or *many2many*.

map setDefault

Sets default values for all configuration options.

map show

Displays the current settings of the current map.

EXAMPLES

```
package require IxTclHal

set chassis      1
set fromCard    1
set toCard      2

map setDefault
map config -type one2many

map new -type one2many
map add $chassis $fromCard 1      $chassis $toCard 1
map add $chassis $fromCard 1      $chassis $toCard 2

map add $chassis $fromCard 2      $chassis $toCard 3
map add $chassis $fromCard 3      $chassis $toCard 4

map show
```

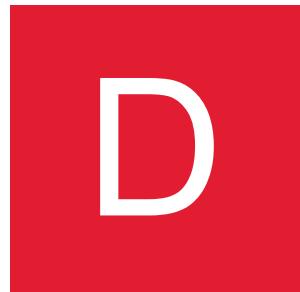
INTERNAL COMMANDS

The following commands are internal interfaces, for use only by Ixia. Use of these commands may produce undesirable results and are not guaranteed to be backward compatible in future releases:

`exists`, `getHelp`, `getType`, `getValidRange`, `getValidValues`, `getValidateProc`

SEE ALSO

getAllPorts, *getRxPorts*, *getTxPorts*



IxTcl Server Usage

IxTcl Server

The IxTcl Server is a software module which implements an intermediate process needed to support non-Windows (Unix) ScriptMate, Tcl and other clients. The version of IxTcl Server must match the version of ScriptMate, Tcl or other clients.

It may either reside on an Ixia chassis or on an intermediate Windows based system between the Unix system and the Ixia chassis. [Figure D-1](#) on page D-2 illustrates the former case.

[Figure D-2](#) on page D-2 illustrates the latter case.

This last case has the advantage that TclServer runs on a different processor than the chassis itself – allowing the chassis to run faster.

Installation and Invocation

IxTcl Server is installed on an Ixia chassis or Windows host using the standard Ixia installation methods. See the *Ixia Quick Start Guide* for a further discussion. The IxTcl Server is listed among the optional components.

When IxTcl Server is installed, it is automatically included in the *All Users Startup* group so that the IxTcl Server will automatically start up when any user logs in. If it is necessary to restart IxTcl Server, then the icon which has been placed on the desktop can be used. The icon is shown below.

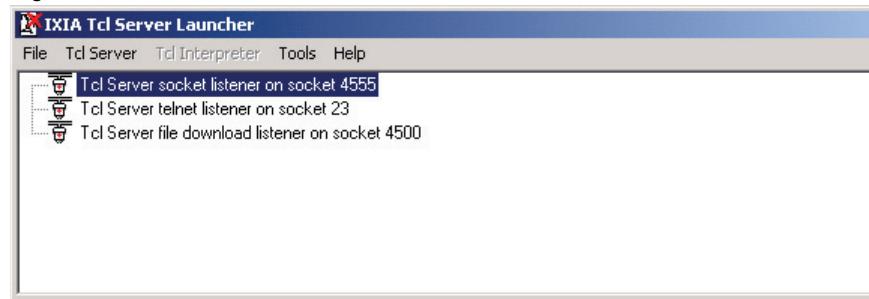


IxTcl Server Usage

Normally IxTcl Server requires no user interaction. In day-to-day usage, it may be safely minimized.

Several options, however, are available for troubleshooting. The initial IxTcl Server screen, before any connections from any clients, is shown in Figure D-1 on page D-2.

Figure D-1. Initial IxTcl Server Screen

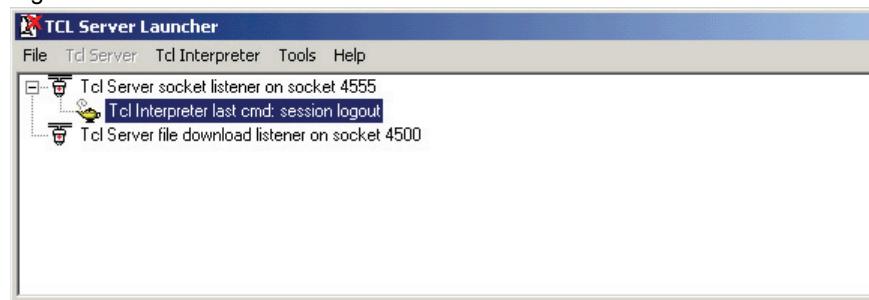


The two lines in the main window indicate that the server is ‘listening’ for connections from clients on two ports:

- Port 4555—this is the default port used by ScriptMate and by Tcl programs which use the Ixia Tcl APIs. All standard connections will be visible in the tree beneath this node.
- Port 4500—this port is used internally by several Ixia products for rapid file transfer.

Connections are reflected within the tree once they have been made as shown in Figure D-2 on page D-2.

Figure D-2. IxTcl Server with Connection



The menus available in this window are:

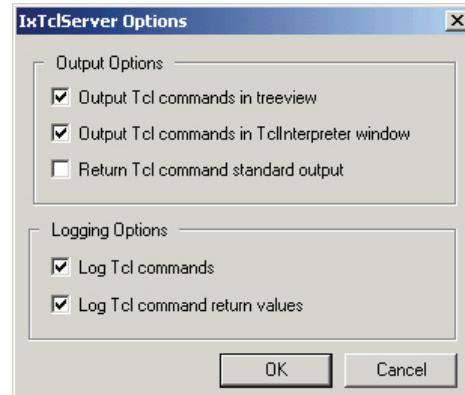
Table D-1. IxTcl Server Menus

Menu	Usage
File	Contains a single 'exit' option.
IxTcl Server	Allows for the creation of additional ports on which IxTcl Server will listen for connections. See Advanced Usage on page D-5 for a further discussion.
Tcl Interpreter	The <i>Show</i> option opens a separate window which displays the commands that are sent through IxTcl Server as well as the results received from the chassis. The contents of this screen are controlled by the <i>Tools..Options</i> menu. This menu option is only active when a IxTcl Server connection is selected. The same window may be opened by right-clicking on a connection and choosing <i>Show</i> .
Tools	Contains a single Options dialog, discussed in Advanced Usage on page D-5.
Help	Contains a single choice 'About IxTcl Server...'. When invoked, a dialog is presented with the version number of IxTcl Server.

Options

The options available with IxTcl Server available by selecting *Tools..Options* from the IxTcl Server window. The dialog is shown in [Figure D-3](#) on page D-3.

Figure D-3. IxTcl Server Options



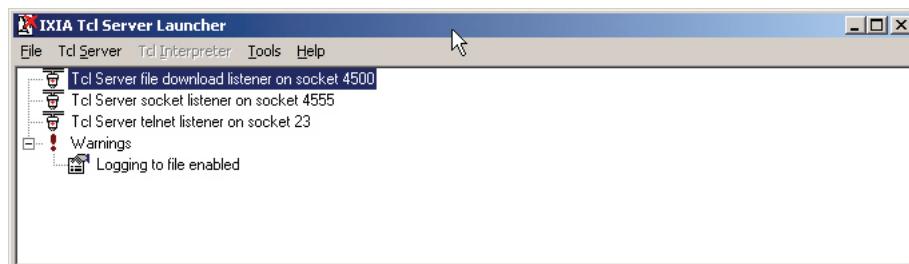
The options available in this dialog are:

Table D-2. IxTcl Server Options

Category	Option	Usage
Output Options	Output Tcl commands in treeview	The last Tcl command executed for a IxTcl Server connection is shown in the tree view. For example, in Figure D-2 on page D-2 the phrase: <i>Tcl Interpreter last cmd: session logout.</i>
	Output Tcl commands in TclInterpreter window	If a Tcl Interpreter window has been opened with the <i>Tcl Interpreter..Show</i> menu choice, then this option indicates that Tcl commands passed through IxTcl Server should be displayed in this window.
	Return Tcl command standard output	See Advanced Usage on page D-5 for a description of this option.
Logging Options	Log Tcl commands	If selected, then a log file is created in the Ixia installation directory (usually <i>C:\Program Files\Ixia</i>). Each connection creates a separate log file whose name includes the year, month, day and seconds since midnight.
	Log Tcl command return values	If selected, return values from the Tcl commands are included in the log.

NOTE: If either of the Logging Options is enabled, additional CPU time is consumed creating and saving the logged data. This may slow down the execution of your test. A warning is added to the IxTcl Server window when one of these options is turned on, as shown in IxTcl Server Windows with Logging Enabled.

Figure D-4. IxTcl Server Windows with Logging Enabled



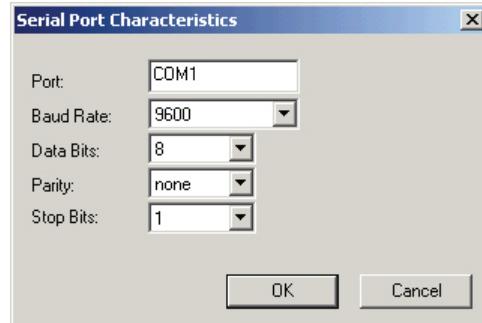
Advanced Usage

Additional means of connecting to IxTcl Server are provided through the *IxTcl Server* menu. The options for this menu choice are:

Table D-3. IxTcl Server Menu Options

Option	Usage
Add Socket Listener...	IxTcl Server's socket interface can 'listen' to ports other than the default port 4555. This option adds another listener at another port.
Add Serial Listener...	IxTcl Server can also listen on one of the host's communications ports. The options associated with this type of connection are shown in Figure D-5 on page D-5. The port should be one of the available ports: COM1, COM2, etc. that is not in use by another application. Specify the port as COMx where x is the port number.
Add Telnet Listener...	IxTcl Server can also listen on a port using the Telnet protocol. A dialog allows the port to be selected.
Delete	This option deletes the currently selected connection in the main window and all associated Tcl Interpreters shown as child nodes

Figure D-5. Serial Port Characteristics



The serial connection and telnet connections are different from the socket listener connection in that they 'speak' Tcl. That is, an external program of any type may make a serial or telnet connection to IxTcl Server and send it Ixia Tcl commands which will be executed on the Ixia chassis. The results of the commands' execution are sent back to the external program. If the *Return Tcl command standard output* option was checked in the Tools..Options dialog, then any output that the command produced would also be sent to the external program.

E

Reserved Keywords

This chapter provides the keywords that are used in IxOS setup. These keywords should not be used as variable names in customer scripts, failing which they will conflict with code execution and exhibit unwanted behavior.

The keywords are listed as follows:

- bgpOriginIGP
- bgpOriginEGP
- bgpOriginIncomplete
- bgpRouteAsPathNoInclude
- bgpRouteAsPathIncludeAsSeq
- bgpRouteAsPathIncludeAsSet
- bgpRouteAsPathIncludeAsSeqConf
- bgpRouteAsPathIncludeAsSetConf
- bgpRouteAsPathPrependAs
- bgpRouteNextHopSetManually
- bgpRouteNextHopSetSameAsLocalIp
- bgpRouteNextHopFixed
- bgpRouteNextHopIncrement
- bgpRouteNextHopIncrementPerPrefix
- bgpCommunityNoExport
- bgpCommunityNoAdvertise
- bgpCommunityExportSubconfed
- bgpSegmentUnknown
- bgpSegmentAsSet
- bgpSegmentAsSequence
- bgpSegmentAsConfedSet
- bgpSegmentAsConfedSequence
- bgpOurIP
- bgpPeerIP
- bgpOurAS
- bgpPeerAS
- bgpOurId
- bgpPeerId
- bgpPeerHoldTimer
- bgpHoldTimer
- bgpMessageSent

- bgpMessageReceived
- bgpUpdateSent
- bgpUpdateReceived
- bgpRoutesAdvertised
- bgpRoutesWithdrawn
- bgpRoutesAdvertisedReceived
- bgpRoutesWithdrawnReceived
- bgpRoutesPerSecondSent
- bgpRoutesPerSecondReceived
- bgpOpenSent
- bgpOpenReceived
- bgpKeepAliveSent
- bgpKeepAliveReceived
- bgpNotificationSent
- bgpNotificationReceived
- bgpCeaseSent
- bgpCeaseReceived
- bgpStateMachineErrorSent
- bgpStateMachineErrorReceived
- bgpHoldTimeExpiredSent
- bgpHoldTimeExpiredReceived
- bgpInvalidOpenSent
- bgpInvalidOpenReceived
- bgpLastErrorReceived
- bgpLastErrorSent
- bgpInvalidOpenUnsupportVersion
- bgpInvalidOpenUnsupportVersion
- bgpInvalidOpenBadPeerAS
- bgpInvalidOpenBadBGPId
- bgpInvalidOpenUnsupportParm
- bgpInvalidOpenAuthenticationFail
- bgpInvalidOpenUnacceptHoldTime
- bgpInvalidOpenSubUnspecified
- bgpUpdateErrorSent
- bgpUpdateErrorReceived
- bgpUpdateErrorAttribListError
- bgpUpdateErrorUnknownWellKnownAttrib
- bgpUpdateErrorMissingWellKnownAttrib
- bgpUpdateErrorAttribFlagError
- bgpUpdateErrorAttribLengthError
- bgpUpdateErrorOriginAttribInvalid
- bgpUpdateErrorASRoutingLoop
- bgpUpdateErrorNextHopAttribInvalid
- bgpUpdateErrorOptionalAttribError
- bgpUpdateErrorNetworkFieldInvalid
- bgpUpdateErrorAsPathInvalid
- bgpUpdateErrorSubUnspecified
- bgpHeaderErrorSent
- bgpHeaderErrorReceived
- bgpHeaderErrorConnNotSyncron
- bgpHeaderErrorBadMsgLength
- bgpHeaderErrorBadMsgType

- bgpHeaderErrorSubUnspecified
- bgpUnspecifiedErrorSent
- bgpUnspecifiedErrorReceived
- bgpActiveOn
- bgpStartsOccured
- bgpStateMachineState
- bgpExternalConnectsReceived
- bgpExternalConnectsAccepted
- bgpGracefulRestartsAttempted
- bgpGracefulRestartsFailed
- bgpRoutesReceivedBeforeStaleTimerExpired
- ospfRouteOriginArea
- ospfRouteOriginExternal
- ospfRouteOriginExternalType2
- ospfRouteOriginNSSA
- ospfRouteOriginSameArea
- ospfOptionBitTypeOfService
- ospfOptionBitExternalRouting
- ospfOptionBitMulticast
- ospfOptionBitNSSACapability
- ospfOptionBitExternalAttributes
- ospfOptionBitDemandCircuit
- ospfOptionBitLSANoForward
- ospfOptionBitUnused
- ospfBroadcast
- ospfPointToPoint
- ospfPointToMultipoint
- ospfInterfaceLinkPointToPoint
- ospfInterfaceLinkTransit
- ospfInterfaceLinkStub
- ospfInterfaceAuthenticationNull
- ospfInterfaceAuthenticationPassword
- ospfInterfaceAuthenticationMD5
- ospfLinkPointToPoint
- ospfLinkTransit
- ospfLinkStub
- ospfLinkVirtual,
- ospfTlvLinkPointToPoint
- ospfTlvLinkMultiAccess
- ospfBBit
- ospfEBit
- ospfVBit
- ospfExternalMetricType1
- ospfExternalMetricType2
- ospfRouterTlv
- ospfLinkTlv
- ospfLsaRouter
- ospfLsaNetwork
- ospfLsaSummaryIp
- ospfLsaSummaryAs
- ospfLsaExternal
- ospfLsaOpaqueLocal

- ospfLsaOpaqueArea
- ospfLsaOpaqueDomain
- ospfLsaNSSA
- isisL3Routing
- dceIsis
- spbIsis
- trillIsis
- isisNormalRouter
- isisRestartingRouter
- isisStartingRouter
- isisHelperRouter
- isisDraftVersion3
- isisDraftVersion4
- isisAuthTypeNone
- isisAuthTypePassword
- isisAuthTypeMD5
- fullyMeshedMapping
- oneToOneMapping
- manualMapping
- isisRouteInternal
- isisRouteExternal
- isisBroadcast
- isisPointToPoint
- isisLevel1
- isisLevel2
- isisLevel1Level2
- isisVLANTypeSingleVLAN
- isisVLANTypeStackedVLANQinQ
- isisUnicast
- isisMulticast
- isisType1
- isisType2
- isisType3
- isisType4
- isisType5
- isisType6
- isisType7
- isisType8
- isisType9
- isisType10
- isisType11
- isisType12
- isisType13
- isisType14
- isisType15
- isisType16
- isisECTAlgorithmC201
- isisECTAlgorithmC202
- isisECTAlgorithmC203
- isisECTAlgorithmC204
- isisECTAlgorithmC205
- isisECTAlgorithmC206

- isisECTAlgorithmC207
- isisECTAlgorithmC209
- isisECTAlgorithmC210
- isisECTAlgorithmC211
- isisTransmissionTypeUnicast
- isisTransmissionTypeMulticast
- isisNodeVLANTypeSingleVLAN
- isisNodeVLANTypeStackedVLANQinQ isisNodeVLANTypeStackedV-LANQinQ
- rsvpNone
- rsvpPrependLoose
- rsvpPrependStrict
- rsvpIngress
- rsvpEgress
- rsvpTrafficEndPoint
- rsvpTunnelEndPoint
- rsvpFF
- rsvpSE
- rsvpEgressAlwaysUseConfiguredStyle
- rsvpEgressAlwaysUseConfiguredStyle
- rsvpEgressUseSEIfInAttribute
- rsvpLabelValueExplicitNull
- rsvpLabelValueRouterAlert
- rsvpLabelValueIPv6ExplicitNull
- rsvpLabelValueImplicitNull
- rsvpTrafficEndpoint
- rsvpTunnelEndpoint
- rsvpP2MPTunnelEndPoint
- rsvpEroIpV4
- rsvpAs
- rsvpRroIpV4
- rsvpLabel
- ripMulticast
- ripBroadcastV1
- ripBroadcastV2
- ripInvalidVersion
- ripReceiveVersion1
- ripReceiveVersion2
- ripReceiveVersion1And2
- ripDefault
- ripSplitHorizon
- ripPoisonReverse
- ripSplitHorizonSpaceSaver
- ripSilent
- ospfNetworkRangeLinkBroadcast
- ospfNetworkRangeLinkPointToPoint
- ospfNetworkRangeLinkPointToPoint
- ldpInterfaceDownstreamUnsolicited
- ldpInterfaceDownstreamOnDemand
- ldpInterfaceDownstreamOnDemand
- ldpInterfaceIndependent
- ldpInterfaceBasic

- ldpInterfaceExtended
- ldpInterfaceExtendedMartini
- atmVcUnidirectional
- atmVcBidirectional
- ldpInterfaceNULL
- ldpInterfaceMD5
- ldpAdvertiseFecRangeNone
- ldpAdvertiseFecRangeFixed
- ldpAdvertiseFecRangeIncrement
- ldpTargetPeerNULL
- ldpTargetPeerMD5
- l2VpnInterfaceFrameRelay
- l2VpnInterfaceATMAAL5
- l2VpnInterfaceATMXCell
- l2VpnInterfaceVLAN
- l2VpnInterfaceEthernet
- l2VpnInterfaceHDLC
- l2VpnInterfacePPP
- l2VpnInterfaceCEM
- l2VpnInterfaceATMVCC
- l2VpnInterfaceATMVPC
- l2VpnInterfaceEthernetVPLS
- l2VpnInterfaceCEIP
- l2VpnInterfaceSatopE1
- l2VpnInterfaceSatopT1
- l2VpnInterfaceSatopE3
- l2VpnInterfaceSatopT3
- l2VpnInterfaceCesoPsnBasic
- l2VpnInterfaceCesoPsnCas
- l2VpnInterfaceFrameRelayRFC4619
- l2VpnInterfaceFrameRelayRFC4619
- ldpL2VpnVcFixedLabel
- ldpL2VpnVcIncrementLabel
- ldpL2VpnPwIdFec
- ldpL2VpnGeneralizedIdFecVpls
- ldpL2VpnVcModelManualConfiguration
- ldpL2VpnVcModelManualConfiguration
- ldpL2VpnVcModelBgpAutoDiscovery
- ldpL2VpnVcModelBgpAutoDiscovery
- ldpL2VpnVcVplsTypeAsNumber
- ldpL2VpnVcVplsTypeIpAddress
- ldpL2VpnVcSrcAiiTypeNumber
- ldpL2VpnVcSrcAiiTypeIpAddress
- ldpL2VpnVcTargetAiiTypeNumber
- ldpL2VpnVcTargetAiiTypeNumber
- ldpL2VpnVcTargetAiiTypeIpAddress
- ldpL2VpnVcTargetAiiTypeIpAddress
- ldpL2VpnVcAbsolute
- ldpL2VpnVcDifferential
- ldpL2VpnVcE1Trunk
- ldpL2VpnVcT1ESFTrunk
- ldpL2VpnVcT1SFTrunk

- ldpL2VpnVcHexVal1
- ldpL2VpnVcHexVal2
- ldpL2VpnVcHexVal3
- ldpL2VpnVcHexVal4
- ldpMulticstP2MPLSP
- ldpMulticstMP2MPLSP
- bgp4NeighborInternal
- bgp4NeighborExternal
- bgp4AsNumModeFixed
- bgp4AsNumModeIncrement
- bgp4MD5", (long) bgp4MD5, 0, 0, 0}
- bgp4NULL", (long) bgp4NULL, 0, 0, 0}
- kTunnelTypePimGreRosenDraft
- kTunnelTypeRSVPP2MP
- kTunnelTypeMLDPP2MP
- bgp4VpnFixedLabel
- bgp4VpnIncrementLabel
- vpnDistinguisherIncrementGlobalPart
- vpnDistinguisherIncrementGlobalPart
- vpnDistinguisherIncrementLocalPart
- vpnDistinguisherIncrementLocalPart
- bgp4DistinguisherTypeAS
- bgp4DistinguisherTypeAS
- bgp4DistinguisherTypeIP
- bgp4Distinguisher2OctetMaxAssignedNumber
- bgp4Distinguisher2OctetMaxAssignedNumber
- bgp4TargetTypeAS
- bgp4TargetTypeIP
- bgp4UmhSelectionFixedLabel
- bgp4UmhSelectionIncrementLabel
- umhSelectionDistinguisherIncrementGlobalPart
- umhSelectionDistinguisherIncrementGlobalPart
- umhSelectionDistinguisherIncrementLocalPart
- umhSelectionDistinguisherIncrementLocalPart
- sourceTreeJoin
- sharedTreeJoin
- recvAddressFamilyIpv4
- recvAddressFamilyIpv6
- recvFullyMeshed
- recvOneToOne
- recvAsDistinguisherType
- recvIpDistinguisherType
- recvAs4BytesDistinguisherType
- kTunnelTypeMulticastRSVPP2MP
- kTunnelTypeMulticastMLDPP2MP
- kTunnelTypeMulticastMLDPP2MP
- sndrAddressFamilyIpv4
- sndrAddressFamilyIpv6
- sndrFullyMeshed
- sndrOneToOne
- sndrDistinguisherTypeAs
- sndrDistinguisherTypeIp

- sndrDistinguisherTypeAs4Byte
- bgpAdVplsRtTypeAsNumber
- bgpAdVplsRtTypeIpAddress
- bgpAdVplsVplsIdTypeAsNumber
- bgpAdVplsVplsIdTypeIpAddress
- bgpAdVplsRdTypeAsNumber
- bgpAdVplsRdTypeIpAddress
- bgpAdVplsVsiIdPeAddress
- bgpAdVplsVsiIdAssignedNumber
- ripngIgnore
- ripngStore
- ripngSplitHorizon
- ripngNoSplitHorizon
- ripngPoisonReverse
- bgp4FamilyInvalidId
- bgp4FamilyIpV4Unicast
- bgp4FamilyIpV4Multicast
- bgp4FamilyIpV4Mpls
- bgp4FamilyIpV4MplsVpn
- bgp4FamilyIpV6Unicast
- bgp4FamilyIpV6Multicast
- bgp4FamilyIpV6Mpls
- bgp4FamilyIpV6MplsVpn
- bgp4FamilyUserDefined
- bgp4FamilyIpVpls
- bgp4FamilyIpV4MulticastVpn
- bgp4FamilyIpV6MulticastVpn
- bgp4FamilyIpAdVpls
- bgp4FamilyIpV4MulticastMplsVpn
- bgp4FamilyIpV4MulticastMplsVpn
- bgp4FamilyIpV6MulticastMplsVpn
- bgp4FamilyIpV6MulticastMplsVpn
- mldQuerierVersion1
- mldQuerierVersion2
- mldVersion1
- mldVersion2
- MLD_GROUPMODE_INCLUDE
- MLD_GROUPMODE_EXCLUDE
- MLD_GROUPMODE_EXCLUDE
- multicastSourceModeInclude
- multicastSourceModeExclude
- igmpHostVersion1
- igmpHostVersion2
- igmpHostVersion3
- igmpVersion1
- igmpVersion2
- igmpVersion3
- igmpQuerierVersion1
- igmpQuerierVersion2
- igmpQuerierVersion3
- INCLUDE
- EXCLUDE

- IGMPV1
- IGMPV2
- IGMPV3
- softwareRestart
- softwareReloadOrUpgrade
- switchToRedundantControlProcessor
- switchToRedundantControlProcessor
- unknown
- ospfv3InterfaceOptionDCBit
- ospfv3InterfaceOptionRBit
- ospfv3InterfaceOptionNBit
- ospfv3InterfaceOptionMCBit
- ospfv3InterfaceOptionEBit
- ospfv3InterfaceOptionV6Bit
- ospfv3InterfacePointToPoint
- ospfv3InterfaceBroadcast
- ospfv3RouteOriginAnotherArea
- ospfv3RouteOriginExternalType1
- ospfv3RouteOriginExternalType2
- ospfv3RouteOriginSameArea
- unicastAddress
- multicastAddress
- ospfv3LsaRouter
- ospfv3LsaNetwork
- ospfv3LsaInterAreaPrefix
- ospfv3LsaInterAreaRouter
- ospfv3LsaAsExternal
- ospfv3LsaLink
- ospfv3LsaIntraAreaPrefix
- ospfv3LsaOptionV6Bit
- ospfv3LsaOptionEBit
- ospfv3LsaOptionMCBit
- ospfv3LsaOptionNBit
- ospfv3LsaOptionRBit
- ospfv3LsaOptionDCBit
- ospfv3PrefixOptionPBit
- ospfv3PrefixOptionMCBit
- ospfv3PrefixOptionLAbit
- ospfv3PrefixOptionNUBit
- ospfv3LsaRouterInterfacePointToPoint
- ospfv3LsaRouterInterfacePointToPoint
- ospfv3LsaRouterInterfaceTransit
- ospfv3LsaRouterInterfaceVirtual
- ospfv3NetworkRangeLinkBroadcast
- ospfv3NetworkRangeLinkBroadcast
- ospfv3NetworkRangeLinkPointToPoint
- pimsmNoDataMdt
- pimsmDataMdtIpv4
- pimsmGenerationIdModeIncremental
- pimsmGenerationIdModeIncremental
- pimsmGenerationIdModeRandom
- pimsmGenerationIdModeConstant

- pimsmMappingFullyMeshed
- pimsmMappingOneToOne
- pimsmJoinsPrunesTypeRP
- pimsmJoinsPrunesTypeG
- pimsmJoinsPrunesTypeSG
- pimsmJoinsPrunesTypeSPTSwitchOver
- pimsmJoinsPrunesTypeSPTSwitchOver
- pimsmJoinsPrunesTypeRegisterTriggeredSG
- pimsmJoinsPrunesTypeRegisterTriggeredSG
- pimsmCRPMeshingTypeFull
- pimsmCRPMeshingTypeOneToOne
- pimsmCRPPriorityTypeSame
- pimsmCRPPriorityTypeIncremental
- pimsmCRPPriorityTypeRandom
- pimsmAll
- pimsmFromSource
- pimsmFromGroup
- isisGridLinkPointToPoint
- isisGridLinkBroadcast
- protocolServerStreamReplace
- protocolServerStreamAppend
- addressTypeIpV4
- addressTypeIpV6
- transmitIgmpJoin
- startIgmp
- transmitIgmpLeave
- startBgp4
- stopBgp4
- startOspf
- stopOspf
- startIsis
- stopIsis
- startRsvp
- stopRsvp
- startRip
- stopRip
- startLdp
- stopLdp
- startRipng
- stopRipng
- startPimsm
- stopPimsm
- startMld
- stopMld
- startOspfv3
- stopOspfv3
- stopIgmp
- startStp
- stopStp
- startEigrp
- stopEigrp
- startBfd

- stopBfd
- startCfm
- stopCfm
- startLacp
- stopLacp
- startOam
- stopOam
- startMplsTp
- stopMplsTp
- startMplsOam
- stopMplsOam
- startElmi
- stopElmi
- igmpReportToOneWhenQueried
- igmpReportToAllWhenQueried
- igmpReportToAllUnsolicited
- stpInterfacePointToPoint
- stpInterfaceShared
- bridgeStp
- bridgeRstp
- bridgeMstp
- bridgePvst
- bridgeRpvst
- bridgePvstp
- stp
- rstp
- bridges
- providerBridges
- stpInterfaceRoleDisabled
- stpInterfaceRoleRoot
- stpInterfaceRoleDesignated
- stpInterfaceRoleAlternate
- stpInterfaceRoleBackup
- stpInterfaceStateDiscarding
- stpInterfaceStateLearning
- stpInterfaceStateForwarding
- eigrpIGRP
- eigrpEnhancedIGRP
- eigrpStatic
- eigrpRIP
- eigrpHelloeigrpOSPF", (long) eigrpOSPF, 0, 0, 0},
- eigrpISIS
- eigrpEGP
- eigrpBGP
- eigrpIDRP
- eigrpConnected
- eigrpExternalRoute
- eigrpCandidateDefault
- eigrpExternal
- eigrpInternal
- bfd1HopSess
- bfdMultihopSess

- kDetectMultiplierMin
- kDetectMultiplierMax
- kMinDesiredMinRxIntv
- kMinDesiredTxIntv
- kMinEchoRxIntv
- kMinEchoTxIntv
- kMinEchoTimeOutIntv
- cfmMIP
- cfmMEP
- cfmPrimaryVid
- cfmCharacterString
- cfmTwoOctet
- cfmRfc2685VpnId
- cfmIccBasedFormat
- cci3msec
- cci10msec
- cci100msec
- cci1sec
- cci10sec
- cci1min
- cci10min
- chassisComponent
- interfaceAlias
- portComponent
- chassisMacAddress
- networkAddress
- interfaceName
- locallyAssigned
- rdiModeAuto
- rdiModeOn
- rdiModeOff
- dmMethodTwoWay
- dmMethodOneWay
- aisModeAuto", (long) aisModeAuto, 0, 0, 0},
- aisModeStart", (long) aisModeStart, 0, 0, 0},
- aisModeStop", (long) aisModeStop, 0, 0, 0},
- ais1sec", (long) ais1sec, 0, 0, 0},
- ais1min", (long) ais1min, 0, 0, 0},
- lckModeAuto", (long) lckModeAuto, 0, 0, 0},
- lckModeStart", (long) lckModeStart, 0, 0, 0},
- lckModeStop", (long) lckModeStop, 0, 0, 0},
- lck1sec", (long) lck1sec, 0, 0, 0},
- lck1min", (long) lck1min, 0, 0, 0},
- tstModeStart", (long) tstModeStart, 0, 0, 0},
- tstModeStop", (long) tstModeStop, 0, 0, 0},
- tstPatternNullSignalWithoutCrc32
- tstPatternNullSignalWithCrc32
- tstPatternPrbs2311WithoutCrc32
- tstPatternPrbs2311WithCrc32
- tstTestTypeInService
- tstTestTypeOutOfService
- lmMethodSingleEnded

- lmMethodDualEnded
- cfmBroadCastLink
- cfmPointToPointLink
- singleVlan
- stackedVlan
- cfmNoNamePresent
- cfmDomainNameString
- cfmMACAddressPlus2OctetInt
- cfmMANNameCharString
- cfm
- y1731
- pbbTe
- ethernet
- llcSnap
- oneSec
- oneMin
- noVlanId
- vlanId
- allVlanId
- unicast
- multicast
- allFormats
- primaryVid
- characterString
- twoOctetInteger
- rfc2685VpnId
- dm
- dvm
- zeroMd
- oneMd
- twoMd
- threeMd
- fourMd
- fiveMd
- sixMd
- sevenMd
- allMd
- mepMac
- mepId
- mepMacAll
- mepIdAll
- linkTrace
- loopback
- delayMeasurement
- lossMeasurement
- manual
- oneToOne
- oneToMany
- allToOne
- allToMany
- fastInterval
- slowInterval

- autoInterval
- defaultInterval
- shortTimeOut
- longTimeOut
- autoTimeout
- defaultTimeOut
- active
- passive
- defaultActivity
- fixedMode
- randomMode
- defaultRequestMethod
- markerFrequencyValueMin
- markerFrequencyValueMax
- markerFrequencyValueDefault
- markerFrequencyLowerValueMin
- markerFrequencyLowerValueMax
- markerFrequencyLowerValueDefault
- markerFrequencyLowerValueDefault
- markerFrequencyUpperValueMin
- markerFrequencyUpperValueMax
- markerFrequencyUpperValueDefault
- markerFrequencyUpperValueDefault
- disableFlag
- enableFlag
- defaultFlag
- activeMode
- passiveMode
- single
- periodic
- disableLoopback
- enableLoopback
- noIncrement
- parallelIncrement
- innerFirst
- outerFirst
- icc
- ietf
- apsIetf
- apsY1731
- lsp
- pw
- nestedLspPw
- rangeRoleNone
- rangeRoleWorking
- rangeRoleProtect
- cccvBfdCv
- cccvBfdCc
- cccvY1731
- cccvNone
- alarmTypeIetf

- alarmTypeY1731
- dmTypeIetf
- dmTypeY1731
- lmTypeIetf
- lmTypeY1731
- onePlusOneUnidirectional
- oneIstoOneBidirectional
- onePlusOneBidirectional
- dmTimeFormatIeee
- dmTimeFormatNtp
- lmCounterType32Bit
- lmCounterType64Bit
- srcVplsIdTypeAsNumber
- srcVplsIdTypeIpAddress
- srcVplsIdTypeasNumber4Bytes
- destVplsIdTypeAsNumber
- destVplsIdTypeIpAddress
- destVplsIdTypeasNumber4Bytes
- cccvPauseTriggerOptionTx
- cccvPauseTriggerOptionRx
- cccvPauseTriggerOptionTxRx
- cccvResumeTriggerOptionTx
- cccvResumeTriggerOptionRx
- cccvResumeTriggerOptionTxRx
- apsTriggerTypeClear
- apsTriggerTypeForcedSwitch
- apsTriggerTypeManualSwitchToProtect
- apsTriggerTypeManualSwitchToProtect
- apsTriggerTypeManualSwitchToWorking
- apsTriggerTypeManualSwitchToWorking
- apsTriggerTypeLockout
- apsTriggerTypeExercise
- apsTriggerTypeFreeze
- alarmTriggerTypeIetf
- alarmTriggerTypeY1731
- dmTriggerTypeIetf
- dmTriggerTypeY1731
- counterType32Bit
- counterType64Bit
- alarmTriggerClear
- alarmTriggerStart
- dmModeNoResponseExpected
- dmModeResponseExpected
- dmTriggerTimeFormatIeee
- dmTriggerTimeFormatNtp
- lmTriggerTypeIetf
- lmTriggerTypeY1731,
- lmModeResponseExpected
- lmModeNoResponseExpected
- pwStatusCodePwNotForwarding
- pwStatusCodeLocalAcRxFault
- pwStatusCodeLocalAcTxFault

- pwStatusCodeLocalPsnFacingPwRxFault
- pwStatusCodeLocalPsnFacingPwRxFault
- pwStatusCodeLocalPsnFacingPwTxFault
- pwStatusCodeLocalPsnFacingPwTxFault
- lspPingEncapsulationTypeGach
- lspPingEncapsulationTypeUDPIP_Gach
- lspPingEncapsulationTypeUDPIP_Gach
- lspTraceRouteEncapsulationTypeGach
- lspTraceRouteEncapsulationTypeGach
- lspTraceRouteEncapsulationTypeUDPIP_Gach
- lspTraceRouteEncapsulationTypeUDPIP_Gach
- minRxInterval_10
- minRxInterval_100
- minRxInterval_1000
- minRxInterval_10000
- minRxInterval_3_33
- minRxInterval_60000
- minRxInterval_600000
- minTxInterval_10
- minTxInterval_100
- minTxInterval_1000
- minTxInterval_10000
- minTxInterval_3_33
- minTxInterval_60000
- minTxInterval_600000
- unexpectedMepId
- unexpectedYourDiscriminator
- onDemandCvPadTlv_drop
- onDemandCvPadTlv_copy
- onDemandCvPadTlv_none
- onDemandCvDownstreamAddressType_ipv4Numbered
- onDemandCvDownstreamAddressType_ipv4Numbered
- onDemandCvDownstreamAddressType_ipv4Unnumbered
- onDemandCvDownstreamAddressType_ipv4Unnumbered
- onDemandCvDownstreamAddressType_nonIp
- onDemandCvDownstreamAddressType_nonIp
- doNotReply
- replyViaIpv4Ipv6UdpPacket
- replyViaIpv4Ipv6UdpPacketWithRouterAlert
- replyViaIpv4Ipv6UdpPacketWithRouterAlert
- replyViaApplicationLevelControlChannel
- replyViaApplicationLevelControlChannel
- dropPadTlvFromReply
- copyPadTlvToReply
- controlChannelRouterAlert
- controlChannelPwAch
- bfdCvTypeIpUdp
- bfdCvTypePwAch
- ipv4NumberedDownStreamAddressType
- ipv4NumberedDownStreamAddressType
- ipv4UnNumberedDownStreamAddressType
- ipv4UnNumberedDownStreamAddressType

- ipv6NumberedDownStreamAddressType
- ipv6NumberedDownStreamAddressType
- ipv6UnNumberedDownStreamAddressType
- ipv6UnNumberedDownStreamAddressType
- triggerDropPadTlvFromReply
- triggerCopyPadTlvToReply
- pause
- resume
- resetToNormalReply
- forceReplyCode
- noReturnCode
- malformedEchoRequestReceived
- oneOrMoreOfTheTlvsWasNotUnderstood
- oneOrMoreOfTheTlvsWasNotUnderstood
- replyingRouterIsAnEgressForTheFecAtStackDepthRsc
- replyingRouterIsAnEgressForTheFecAtStackDepthRsc
- replyingRouterHasNoMappingForTheFecAtStackDepthRsc
- replyingRouterHasNoMappingForTheFecAtStackDepthRsc
- downstreamMappingMismatch
- upstreamInterfaceIndexUnknown
- lspPingReserved
- labelSwitchedAtStackDepthRsc
- labelSwitchedButNoMplsForwardingAtStackDepthRsc
- labelSwitchedButNoMplsForwardingAtStackDepthRsc
- mappingForThisFecIsNotTheGivenLabelAtStackDepthRsc
- mappingForThisFecIsNotTheGivenLabelAtStackDepthRsc
- noLabelEntryAtStackDepthRsc", (long) noLabelEntryAtStackDepthRsc
- protocolNotAssociatedWithInterfaceatFecStackDepthRsc
- protocolNotAssociatedWithInterfaceatFecStackDepthRsc
- prematureTerminationOfPingDueToLabelStackShrinkingToSingleLabel
- bfdPduOptionsPause
- bfdPduOptionsResume
- tx
- rx
- txRx
- triggerDoNotReply", (long) triggerDoNotReply
- triggerReplyViaIpv4Ipv6UdpPacket
- triggerReplyViaIpv4Ipv6UdpPacket
- triggerReplyViaIpv4Ipv6UdpPacketWithRouterAlert
- triggerReplyViaIpv4Ipv6UdpPacketWithRouterAlert
- triggerReplyViaApplicationLevelControlChannel
- triggerReplyViaApplicationLevelControlChannel
- ipv4Numbered_downstreamAddressType
- ipv4Numbered_downstreamAddressType
- ipv4UnNumbered_downstreamAddressType
- ipv4UnNumbered_downstreamAddressType
- uniC
- uniN
- elmiAllToOne
- elmiNoBundling
- elmiBundling
- p2p

- mp2mp
- elmiNotActive
- elmiNewNNotActive
- elmiNewNActive
- elmiActiveve
- elmiPartiallyActive
- elmiNewNPartiallyActive

Index

Numerics

10/100 Mii 5-6

10GE 5-10, A-39

A

Advanced Scheduler 5-7

advertise100FullDuplex 5-8

advertise100HalfDuplex 5-8

advertise10FullDuplex 5-8

advertise10HalfDuplex 5-8

advertiseAbilities 5-9

API 4-1

API Structure and Conventions 4-1

aremovedvertise1000Full 5-9

ARP 5-71

arp A-2

associationHeader A-9

ATM 5-29

atmFilter 5-96, A-10

atmHeader 5-31, A-12

atmHeaderCounter 5-31, A-15

atmPort 5-30, A-35

atmReassembly 5-95, A-37

atmStat 5-105, A-40

autoDetectInstrumentation A-45

B

BERT 5-6, 5-7, 5-8

bert 5-27

bertErrorGeneration 5-27, A-59

bertUnframed 5-29, A-62

Bit Error Rate Testing 5-7, 5-8, A-51

broadcastTopology 5-4

byte2IpAddr 3-18, B-2

C

calculateFPS B-3

calculateGapBytes B-4

calculateMaxRate 3-17, B-5

calculatePercentMaxRate B-6

Calculation Utilities 3-17

Capture 5-8

capture 5-92, A-67

Capture Data 3-22

captureBuffer 5-92, A-73

card 5-6, A-79

Cards 5-2

CDMA Server 5-5

cget 4-1

Chassis 3-7, 5-2

chassis 5-5, A-99

chassisChain 5-4, A-107

Cleanup 3-3

cleanUp 3-9

collisionBackoff A-109

conditionalStats A-110

conditionalTable A-114

config 4-1

Console Output 3-26

customOrderedSet 5-40, A-116

D

Data Capture 3-19, 5-89

Data Integrity 5-8

Data Link Layer 5-67

Data Transmission 3-12, 5-49

dataIntegrity 5-63, A-117

DCC 5-7, 5-8, 5-16, 5-54

dcc 5-16, A-122

decode 4-2

dectohex 3-18, B-11

dhcp 5-83, A-125

disableUdfs 3-14, B-12

discoveredAddress A-143

discoveredList 5-113, A-144

discoveredNeighbor A-146

DLL 1-3

E

enableEvents 3-27, B-13

encHeader A-147

errorMsg 3-26, B-14

espHeader A-149

Expect 1-4

extendedLinkServices A-150

F

fcEOF A-156

fcNameServer A-158

fcNameServerQuery A-160
fcoe A-162
fcoeDiscoveredInfo A-174
fcoeNameServer A-176
fcoePlogi A-178
fcoeProperties A-179
fcPlogi A-186
fcPort A-187
fcProperties A-190
fcSOF A-192
fecError A-194
fibreChannel A-199
filter 5-89, A-218
filter config 4-8
filterPallette 5-91, A-228
filterPallette config 4-8
First Time Stamp 5-8
Flows 5-49
forcedCollisions 5-64, A-244
Frame Data 5-56
frameRelay 5-26, A-247

G

General Purpose Commands 3-9
get 4-2
getAllPorts 3-6, C-2
getErrorString B-15
getQueue 5-50
getRxPorts 3-6, C-3
getStatLabel B-16
getTxPorts 3-6, C-4
GPS Server 5-5

H

hdlc 5-25, A-261
hextodec 3-18, B-17
High-Level API 3-1
host2addr 3-18, B-18

I

icmp A-267
icmpV6 A-271
icmpV6Error A-275
icmpV6Informational A-277
icmpV6MulticastListener A-278
icmpV6NeighborDiscovery A-279

icmpV6OptionLinkLayerDestination A-282
icmpV6OptionLinkLayerSource A-283
icmpV6OptionMaxTransmissionUnit A-284
icmpV6OptionPrefixInformation A-285
icmpV6OptionRedirectedHeader A-287
icmpV6OptionUserDefine A-288
icmpV6UserDefine A-289
IFRHeader A-290
igmp 5-80, A-292
Including Source Code 3-7
Initialization 3-3
installing tarball 2-6
Interface Table 5-109, 5-121
interfaceEntry 5-112, A-299
interfaceIpV4 5-113, A-303
interfaceIpV6 5-113, A-304
interfaceTable 5-110, A-305
IP 5-72, 5-119
ip A-320
IP address table 5-121
ip config 4-7
ipAddressTable 5-119, A-331
ipAddressTableItem 5-121, A-333
ipV6 5-74, A-335
ipV6Address A-347
ipV6Authentication 5-76, A-352
ipV6Destination 5-77, A-354
ipV6Fragment 5-77, A-356
ipV6HopByHop 5-77, A-358
ipV6OptionBindingAck A-367
ipV6OptionBindingRequest A-369
ipV6OptionBindingUpdate A-365
ipV6OptionHomeAddress A-368
ipV6OptionJumbo A-363
ipV6OptionMIPv6AlternativeCoaSub A-371
ipV6OptionMIPv6UniqueIdSub A-370
ipV6OptionPAD1 A-361
ipV6OptionPADN A-362
ipV6OptionRouterAlert A-364
ipV6OptionUserDefine A-372
ipV6Routing 5-78, A-373
IPX 5-69
ipx A-375
isl 5-67, A-382
ixCheckLinkState 3-13, C-11

ixCheckOwnership 3-10, C-13
 ixCheckPortTransmitDone 3-16, C-17
 ixCheckPPPState 3-13, C-15
 ixCheckTransmitDone 3-16, C-19
 ixClearArpTable 3-24, C-22
 ixClearOwnership 3-10, C-24
 ixClearPacketGroups 3-21, C-26
 ixClearPortArpTable 3-24, C-30
 ixClearPortPacketGroups 3-21, C-32
 ixClearPortStats 3-21, C-34
 ixClearScheduledTransmitTime 3-17, C-36
 ixClearStats 3-21, C-37
 ixClearTimeStamp 3-21, C-39
 ixCollectStats 3-22, C-41
 ixConnectToChassis 3-8, C-44
 ixConnectToTclServer 3-7, C-47
 ixConvertToSeconds C-48
 ixCreatePortListWildCard 3-5, C-49
 ixCreateSortedPortList 3-6, C-51
 ixDisableArpResponse 3-24
 ixDisablePortArpResponse 3-24
 ixDisconnectFromChassis 3-8, C-56
 ixDisconnectTclServer 3-7, C-57
 ixEnableArpResponse 3-24
 ixEnableIntrinsicLatencyAdjustment C-62
 ixEnablePortArpResponse 3-24
 ixEnablePortIntrinsicLatencyAdjustment C-63
 ixErrorInfo C-64
 ixGetChassisID 3-8, C-65
 ixGetLineUtilization C-66
 ixGlobalSetDefault C-67
 IXIA 100 5-2
 ixIsIntrinsicLatencyAdjustmentEnabled C-70
 ixIsOverlappingIpAddress C-71
 ixIsSameSubnet C-72
 ixIsValidHost C-73
 ixIsValidNetMask C-74
 ixIsValidUnicastIp C-75
 ixLogin 3-10, C-78
 ixLogout 3-10, C-79
 ixMiiConfig C-80
 ixPortClearOwnership 3-10, C-82
 ixPortTakeOwnership 3-10, C-84
 ixProxyConnect 3-7, C-86
 ixPuts 3-26, C-88
 ixRequestStats 3-23, C-89
 ixResetPortSequenceIndex 3-21, C-90
 ixResetSequenceIndex 3-21, C-92
 ixRestartAutoNegotiation 3-15, C-94
 ixRestartPortAutoNegotiation 3-15, C-95
 ixRestartPortPPPAutoNegotiation C-96
 ixRestartPortPPPNegotiation C-96
 ixRestartPPPNegotiation C-97
 IxSampleTcl.tcl 2-7
 ixSetAdvancedStreamSchedulerMode 3-14, C-98
 ixSetAutoDetectInstrumentationMode C-100
 ixSetCaptureMode 3-19, C-101
 ixSetDataIntegrityMode 3-20, C-103
 ixSetPacketFlowMode 3-13, C-105
 ixSetPacketGroupMode 3-20, C-107
 ixSetPacketStreamMode 3-13, C-109
 ixSetPortAdvancedStreamSchedulerMode C-111
 ixSetPortCaptureMode 3-19, C-113
 ixSetPortDataIntegrityMode 3-20, C-115
 ixSetPortPacketFlowMode 3-13, C-117
 ixSetPortPacketGroupMode 3-20, C-119
 ixSetPortPacketStreamMode 3-13, C-121
 ixSetPortSequenceCheckingMode 3-20, C-123
 ixSetPortRoundTripFlowMode 3-14, C-125
 ixSetScheduledTransmitTime 3-17, C-127
 ixSetSequenceCheckingMode 3-20, C-128
 ixSetTcpRoundTripFlowMode 3-14, C-130
 ixSimulatePhysicalInterfaceDown C-132
 ixSimulatePhysicalInterfaceUp C-133
 ixSimulatePortPhysicalInterfaceDown C-134
 ixSimulatePortPhysicalInterfaceUp C-135
 ixSource 3-7, C-136
 ixStartAtmOamTransmit 3-17
 ixStartCapture 3-22, C-138
 ixStartCollisions 3-16, C-140
 ixStartPacketGroups 3-22, C-143
 ixStartPortAtmOamTransmit 3-17
 ixStartPortCapture C-147
 ixStartPortCollisions 3-16, C-149
 ixStartPortPacketGroups 3-22, C-151
 ixStartPortTransmit 3-15, C-154
 ixStartStaggeredTransmit 3-16, C-156
 ixStartTransmit 3-15, C-158
 ixStopAtmOamTransmit 3-17
 ixStopCapture 3-22, C-161

ixStopCollisions 3-16, C-163
ixStopPacketGroups 3-22, C-166
ixStopPortAtmOamTransmit 3-17
ixStopPortCapture 3-22, C-170
ixStopPortCollisions 3-16, C-172
ixStopPortPacketGroups 3-22, C-174
ixStopPortTransmit 3-15, C-177
ixStopTransmit 3-15, C-179
ixTakeOwnership 3-10, C-181
IxTclHAL 1-1
IxTclHal 4-5
ixTransmitArpRequest 3-25, C-183
ixTransmitPortArpRequest 3-25, C-185
ixWriteConfigToHardware 3-9, C-188
ixWritePortsToHardware 3-9, C-190

L

lasi A-385
lcas A-389
Link Fault Signaling 5-39
linkFaultSignaling 5-39, A-391
Logging 3-26
logMsg 3-27, B-19
logOff 3-26, B-20
logOn 3-26, B-21

M

macSecChannel A-396
macSecRx A-398
macSecTag A-400
macSecTx A-404
Many to Many mapping 4-6
Many to One mapping 4-6
many2manyArray 3-5, 4-6, 4-9
manyZoneArray 3-4, 4-6, 4-9
Many-to-many mapping 3-3
Many-to-One mapping 3-3
map 3-5, 4-5, C-191
Mapping 3-3
mapping 4-5
mii A-406
mpls 5-69, A-416
mplsLabel 5-69, A-420

N

networkHeader A-422

npivProperties A-427

O

oamEventNotification A-430
oamEventOrgTlv A-432
oamFramePeriodTlv A-435
oamFrameTlv A-433
oamHeader A-437
oamInformation A-442
oamLocalInformationTlv A-444
oamLoopbackControl A-446
oamOrganizationSpecific A-447
oamOrganizationSpecificTlv A-448
oamPort A-449
oamRemoteInformationTlv A-452
oamStatus A-454
oamSummaryTlv A-456
oamSymbolPeriodTlv A-458
oamVariableRequest A-460
oamVariableRequestTlv A-462
oamVariableResponse A-463
oamVariableResponseTlv A-465
One to Many mapping 4-6
One to One mapping 4-5
one2manyArray 3-4, 4-6, 4-9
one2oneArray 3-4, 4-6, 4-9
One-to-Many mapping 3-3
One-to-One mapping 3-3
opticalDigitalWrapper A-467

P

package require 1-3, 4-5
Packet flow 5-7
Packet over Sonet 5-6, 5-13
Packet stream 5-7
PacketGroup 5-8
packetGroup 5-60, A-469
packetGroupStats A-482
pauseControl 5-84
poeAutoCalibration A-508
poePoweredDevice A-510
poeSignalAcquisition A-518
Port 5-123
port 5-6, A-521
port config 4-7
Port Lists 3-3

Port Ownership 3-10
 portCpu A-555
 portGroup 5-47, A-557
 Ports 5-2
 POS 5-6
 ppp 5-23, A-563
 pppStatus 5-23, A-570
 PRBS packets 5-8
 prbsCapture A-574
 protocol 5-66, A-576
 Protocol Server 5-108
 protocolOffset 5-66, A-580
 protocolPad A-585
 Protocols 5-66
 protocolServer 5-108
 ptp A-586
 ptpAnnounce A-592
 ptpDelayRequest A-595
 ptpDelayResponse A-596
 ptpDiscoveredInfo A-597
 ptpFollowUp A-599
 ptpProperties A-600
 ptpSync A-603

Q

qos 5-95, A-604

R

Reserved keywords E-1
 reserved keywords E-1
 Resilient Packet Ring 5-16
 resourceGroupEx A-93
 rip A-608
 ripRoute 5-82, A-612
 RPR 5-16
 rprFairness 5-18, A-614
 rprOam 5-19, A-622
 rprProtection 5-18, A-625
 rprRingControl 5-17, A-628
 rprTlvBandwidthPair 5-20, A-631
 rprTlvIndividualBandwidth 5-20, A-632
 rprTlvNeighborAddress 5-22, A-634
 rprTlvStationName 5-22, A-636
 rprTlvTotalBandwidth 5-21, A-638
 rprTlvVendorSpecific 5-22, A-640
 rprTlvWeight 5-21, A-642

rprTopology 5-20, A-644

S

Sequence Checking 5-8, 5-64
 sequenceNumberUdf A-647
 session 5-4, A-653
 set 4-1
 setDefault 4-2, 5-10
 setFactoryDefaults 5-10
 setModeDefaults 5-10
 setQueue 5-50
 Setup 3-3
 sfpPlus A-655
 showCmd B-23
 SNTP Server 5-5
 SONET 5-7
 sonet 5-14, A-657
 SONET DCC 5-8
 sonetCircuit A-663
 sonetCircuitList A-665
 sonetCircuitProperties A-672
 sonetError 5-15, A-674
 sonetOverhead 5-16, A-679
 SPE 5-7, 5-54
 splitPacketGroup A-682
 srpArp 5-85, A-685
 srpDiscovery 5-86, A-690
 srpIps 5-85, A-693
 srpMacBinding 5-86, A-696
 srpUsage 5-87, A-697
 Start Transmit 3-15
 startTime 5-4
 stat 5-96, A-703
 statAggregator A-738
 statGroup 5-97, A-741
 Statistics 3-19, 3-22, 5-89
 statList A-743
 statWatch 5-97, A-744
 stream 5-50, A-747
 stream config 4-7
 streamExtractorFilter A-768
 streamExtractorModifier A-773
 streamQueue 5-88, A-775
 streamQueueList 5-87, A-778
 streamRegion A-780

Streams 5-49
streamTransmitStats A-782

T

tableUdf 5-58
Tcl/Tk 1-3
Tcl-DP 1-4
TclServer 3-7
TCLTransceiver A-794
tcp 5-79, A-802
TCP Round Trip 5-7
TCP Round Trips 5-8
tcpRoundTripFlow A-806
tcpRoundTripFlows 5-59
The 5-69, 5-71
timeServer 5-2, 5-4, A-811
txRxPreamble 5-40, A-819

U

udf A-822
UDF Cascade 5-57
udf config 4-8
udp 5-80, A-831
usb A-836
user 3-8, B-24
User Defined Fields 5-56

V

version 5-4, A-838
VFTHeader A-840
vlan 5-67, A-841
vsrError 5-103, A-845
vsrStat 5-103, A-851

W

weightedRandomFramesize 5-55, A-855
Wide packet group 5-8
Wish 1-3
write 4-2

X

xaui 5-13, A-860