



# **Intel® IXP4XX Product Line Access Layer API & Codelets Reference**

Automatically generated from sources, December 12, 2003.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel products are not intended for use in medical, life saving, life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Intel® IXP400 Software may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's website at <http://www.intel.com>.

AlertVIEW, AnyPoint, AppChoice, BoardWatch, BunnyPeople, CablePort, Celeron, Chips, CT Connect, CT Media, Dialogic, DM3, EtherExpress, ETOX, FlashFile, i386, i486, i960, iCOMP, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Create & Share, Intel GigaBlade, Intel InBusiness, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel Play, Intel Play logo, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel TeamStation, Intel Xeon, Intel XScale, IPLink, Itanium, LANDesk, LanRover, MCS, MMX, MMX logo, Optimizer logo, OverDrive, Paragon, PC Dads, PC Parents, PDCharm, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your Command, RemoteExpress, Shiva, SmartDie, Solutions960, Sound Mark, StorageExpress, The Computer Inside., The Journey Inside, TokenExpress, Trillium, VoiceBrick, Vtune, and Xircom are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\*Other names and brands may be claimed as the property of others.

# Table of Contents

<b>IXP425 Operating System Services Library (IxOSSL) API.....</b>	<b>1</b>
Data Structures.....	1
Defines.....	1
Typedefs.....	2
Enumerations.....	3
Functions.....	3
Detailed Description.....	7
Define Documentation.....	7
Typedef Documentation.....	11
Enumeration Type Documentation.....	12
Function Documentation.....	13
<b>IXP425 ADSL Driver API.....</b>	<b>28</b>
Typedefs.....	28
Enumerations.....	28
Functions.....	29
Detailed Description.....	29
Typedef Documentation.....	30
Enumeration Type Documentation.....	30
Function Documentation.....	31
<b>IXP425 Assertion Macros (IxAssert) API.....</b>	<b>39</b>
Defines.....	39
Detailed Description.....	39
Define Documentation.....	39
<b>IXP425 ATM Driver Access (IxAtmdAcc) API.....</b>	<b>40</b>
Defines.....	40
Typedefs.....	41
Enumerations.....	41
Functions.....	42
Detailed Description.....	43
Define Documentation.....	44
Typedef Documentation.....	45
Enumeration Type Documentation.....	48
Function Documentation.....	49
<b>IXP425 ATM Driver Access (IxAtmdAcc) Control API.....</b>	<b>62</b>
Modules.....	62
Defines.....	62
Typedefs.....	62
Functions.....	63
Detailed Description.....	64
Define Documentation.....	64
Typedef Documentation.....	65
Function Documentation.....	69

# Table of Contents

<b>IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API [IXP425 ATM Driver Access (IxAtmdAcc) Control API].....</b>	<b>82</b>
Data Structures.....	82
Detailed Description.....	83
<b>IXP425 ATM Manager (IxAtmm) API.....</b>	<b>84</b>
Data Structures.....	84
Defines.....	84
Typedefs.....	85
Enumerations.....	85
Functions.....	85
Detailed Description.....	87
Define Documentation.....	87
Typedef Documentation.....	88
Enumeration Type Documentation.....	89
Function Documentation.....	90
<b>IXP425 ATM Transmit Scheduler (IxAtmSch) API.....</b>	<b>99</b>
Defines.....	99
Functions.....	99
Detailed Description.....	100
Define Documentation.....	100
Function Documentation.....	101
<b>IXP425 ATM Types (IxAtmTypes).....</b>	<b>107</b>
Data Structures.....	107
Defines.....	107
Typedefs.....	108
Enumerations.....	109
Detailed Description.....	109
Define Documentation.....	110
Typedef Documentation.....	112
Enumeration Type Documentation.....	113
<b>IXP425 DMA Types (IxDmaTypes).....</b>	<b>115</b>
Enumerations.....	115
Detailed Description.....	116
Enumeration Type Documentation.....	116
<b>IXP425 DMA Access Driver (IxDmaAcc) API.....</b>	<b>119</b>
Defines.....	119
Typedefs.....	119
Functions.....	119
Detailed Description.....	119
Define Documentation.....	120
Typedef Documentation.....	120
Function Documentation.....	120

# Table of Contents

<b>IXP425 Ethernet Access (IxEthAcc) API.....</b>	<b>123</b>
Data Structures.....	123
Defines.....	123
Typedefs.....	124
Enumerations.....	124
Functions.....	125
Detailed Description.....	127
Define Documentation.....	127
Typedef Documentation.....	130
Enumeration Type Documentation.....	131
Function Documentation.....	133
<b>IXP425 Ethernet Database (IxEthDB) API.....</b>	<b>155</b>
Data Structures.....	155
Defines.....	155
Typedefs.....	155
Enumerations.....	156
Functions.....	156
Detailed Description.....	157
Define Documentation.....	157
Typedef Documentation.....	158
Enumeration Type Documentation.....	159
Function Documentation.....	159
<b>IXP425 Ethernet Database Port Definitions (IxEthDBPortDefs).....</b>	<b>167</b>
Data Structures.....	167
Defines.....	167
Enumerations.....	167
Detailed Description.....	168
Define Documentation.....	168
Enumeration Type Documentation.....	168
<b>IXP425 Ethernet Phy Access (IxEthMii) API.....</b>	<b>170</b>
Functions.....	170
Detailed Description.....	170
Function Documentation.....	170
<b>IXP425 Ethernet NPE (IxEthNpe) API.....</b>	<b>175</b>
Defines.....	175
Detailed Description.....	179
Define Documentation.....	179
<b>IXP425 Feature Control (IxFeatureCtrl) API.....</b>	<b>190</b>
Modules.....	190
Defines.....	190
Typedefs.....	193
Functions.....	193
Detailed Description.....	194

# Table of Contents

<b>IXP425 Feature Control (IxFeatureCtrl) API</b>	
Define Documentation.....	194
Typedef Documentation.....	201
Function Documentation.....	201
<b>Software Configuration for Access Component [IXP425 Feature Control (IxFeatureCtrl) API].....</b>	<b>205</b>
Defines.....	205
Enumerations.....	205
Detailed Description.....	205
Define Documentation.....	205
Enumeration Type Documentation.....	206
<b>IXP425 HSS Access (IxHssAcc) API.....</b>	<b>207</b>
Data Structures.....	207
Defines.....	207
Typedefs.....	208
Enumerations.....	209
Functions.....	212
Detailed Description.....	214
Define Documentation.....	215
Typedef Documentation.....	217
Enumeration Type Documentation.....	219
Function Documentation.....	226
<b>IXP425 NPE–A (IxNpeA) API.....</b>	<b>236</b>
Data Structures.....	236
Defines.....	236
Enumerations.....	244
Detailed Description.....	244
Define Documentation.....	244
Enumeration Type Documentation.....	262
<b>IXP425 NPE–Downloader (IxNpeDI) API.....</b>	<b>264</b>
Modules.....	264
Data Structures.....	264
Defines.....	264
Typedefs.....	265
Enumerations.....	265
Functions.....	266
Detailed Description.....	267
Define Documentation.....	267
Typedef Documentation.....	271
Enumeration Type Documentation.....	271
Function Documentation.....	272
<b>IXP425 NPE Image ID Definition [IXP425 NPE–Downloader (IxNpeDI) API].....</b>	<b>282</b>
Defines.....	282
Detailed Description.....	283

# Table of Contents

<b>IXP425 NPE Image ID Definition [IXP425 NPE–Downloader (IxNpeDI) API]</b>	
Define Documentation.....	284
<b>IXP425 NPE Message Handler (IxNpeMh) APL.....</b>	<b>290</b>
Data Structures.....	290
Defines.....	290
Typedefs.....	290
Enumerations.....	290
Functions.....	291
Detailed Description.....	291
Define Documentation.....	292
Typedef Documentation.....	292
Enumeration Type Documentation.....	293
Function Documentation.....	293
<b>IXP425 OS Memory Buffer Management (IxOsBuffMgt) APL.....</b>	<b>299</b>
Data Structures.....	299
Defines.....	299
Typedefs.....	300
Detailed Description.....	300
Define Documentation.....	300
<b>IXP425 OS Memory Buffer Pool Management (IxOsBuffPoolMgt) APL.....</b>	<b>302</b>
Data Structures.....	302
Defines.....	302
Typedefs.....	303
Enumerations.....	303
Functions.....	303
Detailed Description.....	304
Define Documentation.....	304
Typedef Documentation.....	308
Enumeration Type Documentation.....	308
Function Documentation.....	309
<b>IXP425 OS Cache MMU (IxOsCacheMMU) APL.....</b>	<b>316</b>
Defines.....	316
Functions.....	316
Detailed Description.....	317
Define Documentation.....	317
Function Documentation.....	321
<b>IXP425 OS Services (IxOsServices) APL.....</b>	<b>323</b>
Defines.....	323
Typedefs.....	324
Enumerations.....	324
Functions.....	324
Detailed Description.....	326
OsServices I/O Memory Allocation and Access Routines.....	326

# Table of Contents

<b>IXP425 OS Services (IxOsServices) API</b>	
Usage Information.....	326
Macros for memory mapped I/O access:.....	327
Macros for sharing data with the NPEs:.....	328
Define Documentation.....	328
Typedef Documentation.....	332
Enumeration Type Documentation.....	332
Function Documentation.....	333
<b>IXP425 Performance Profiling (IxPerfProfAcc) APL.....</b>	<b>340</b>
Data Structures.....	340
Defines.....	340
Enumerations.....	340
Functions.....	345
Detailed Description.....	346
Define Documentation.....	347
Enumeration Type Documentation.....	348
Function Documentation.....	358
<b>IXP425 Queue Manager (IxQMgr) APL.....</b>	<b>369</b>
Data Structures.....	369
Defines.....	369
Typedefs.....	370
Enumerations.....	371
Functions.....	373
Variables.....	376
Detailed Description.....	376
Define Documentation.....	376
Typedef Documentation.....	381
Enumeration Type Documentation.....	382
Function Documentation.....	384
<b>IXP425 Timer Control (IxTimerCtrl) APL.....</b>	<b>401</b>
Defines.....	401
Typedefs.....	401
Enumerations.....	401
Functions.....	401
Detailed Description.....	402
Define Documentation.....	402
Typedef Documentation.....	402
Enumeration Type Documentation.....	403
Function Documentation.....	403
<b>IXP425 Types (IxTypes).....</b>	<b>406</b>
Defines.....	406
Typedefs.....	406
Detailed Description.....	407
Define Documentation.....	407



# Table of Contents

<b>IXP425 UART Access (IxUARTAcc) APL.....</b>	<b>408</b>
Modules.....	408
Data Structures.....	408
Enumerations.....	408
Functions.....	408
Detailed Description.....	409
Enumeration Type Documentation.....	409
Function Documentation.....	409
<b>Defines for Default Values [IXP425 UART Access (IxUARTAcc) API].....</b>	<b>412</b>
Defines.....	412
Detailed Description.....	412
Define Documentation.....	412
<b>Defines for IOCTL Commands [IXP425 UART Access (IxUARTAcc) API].....</b>	<b>414</b>
Defines.....	414
Detailed Description.....	414
Define Documentation.....	414
<b>Defines for IOCTL Arguments [IXP425 UART Access (IxUARTAcc) API].....</b>	<b>416</b>
Defines.....	416
Detailed Description.....	416
Define Documentation.....	417
<b>IXP400 Version ID (IxVersionId).....</b>	<b>419</b>
Defines.....	419
Detailed Description.....	419
Define Documentation.....	419
<b>IXP425 USB Driver Public APL.....</b>	<b>421</b>
Data Structures.....	421
Defines.....	421
Typedefs.....	426
Enumerations.....	426
Functions.....	428
Detailed Description.....	430
Define Documentation.....	430
Enumeration Type Documentation.....	443
Function Documentation.....	445
<b>IXP425 Codelets.....</b>	<b>451</b>
Modules.....	451
Detailed Description.....	451
<b>IXP425 ATM Codelet (IxAtmCodelet) API [IXP425 Codelets].....</b>	<b>452</b>
Defines.....	452
Functions.....	452
Detailed Description.....	452

# Table of Contents

<b>IXP425 ATM Codelet (IxAtmCodelet) API [IXP425 Codelets]</b>	
Define Documentation.....	457
Function Documentation.....	458
<b>IXP425 DMA Access Codelet (IxDmaAccCodelet) API [IXP425 Codelets].....</b>	<b>459</b>
Defines.....	459
Functions.....	459
Detailed Description.....	459
Define Documentation.....	460
Function Documentation.....	461
<b>IXP425 Ethernet Aal5 (IxEthAal5App) API [IXP425 Codelets].....</b>	<b>462</b>
Defines.....	462
Functions.....	463
Detailed Description.....	463
Define Documentation.....	466
Function Documentation.....	468
<b>IXP425 Ethernet Access Codelet (IxEthAccCodelet) API [IXP425 Codelets].....</b>	<b>470</b>
Defines.....	470
Functions.....	471
Detailed Description.....	471
Define Documentation.....	473
Function Documentation.....	475
<b>IXP425 HSS Access Codelet (IxHssAccCodelet) API [IXP425 Codelets].....</b>	<b>476</b>
Defines.....	476
Functions.....	476
Detailed Description.....	476
<b>IXP425 PerfProf Access Codelet [IXP425 Codelets].....</b>	<b>479</b>
Defines.....	479
Enumerations.....	479
Functions.....	479
Detailed Description.....	479
Define Documentation.....	482
Enumeration Type Documentation.....	482
<b>IXP425 Timers (IxTimersCodelet) API [IXP425 Codelets].....</b>	<b>484</b>
Typedefs.....	484
Enumerations.....	484
Functions.....	484
Detailed Description.....	485
Typedef Documentation.....	485
Enumeration Type Documentation.....	485
Function Documentation.....	486

# Table of Contents

<b>IXP425 USB RNDIS Codelet (IxUSBRNDIS) API [IXP425 Codelets].....</b>	<b>490</b>
Modules.....	490
Functions.....	490
Detailed Description.....	491
Function Documentation.....	491
<b>IXP425 USB RNDIS End Driver Codelet (IxUSBRNDIS) API [IXP425 USB RNDIS Codelet (IxUSBRNDIS) API].....</b>	<b>495</b>
Defines.....	495
Functions.....	495
Detailed Description.....	495
Define Documentation.....	498
Function Documentation.....	498
<b>IXP425 USB RNDIS Vendor Codelet (IxUSBRNDIS) [IXP425 USB RNDIS Codelet (IxUSBRNDIS) API].....</b>	<b>500</b>
Defines.....	500
Detailed Description.....	500
Define Documentation.....	500
<b>ChannelisedStats Struct Reference.....</b>	<b>502</b>
Data Fields.....	502
Detailed Description.....	502
<b>GeneralStats Struct Reference.....</b>	<b>503</b>
Data Fields.....	503
Detailed Description.....	503
<b>ix_ossl_thread_main_info_t Struct Reference [IXP425 Operating System Services Library (IxOSSL) API].....</b>	<b>504</b>
Data Fields.....	504
Detailed Description.....	504
Field Documentation.....	504
<b>ix_ossl_time_t Struct Reference [IXP425 Operating System Services Library (IxOSSL) API].....</b>	<b>505</b>
Data Fields.....	505
Detailed Description.....	505
Field Documentation.....	505
<b>IxAtmCodeletStats Struct Reference.....</b>	<b>506</b>
Data Fields.....	506
Detailed Description.....	506
<b>IxAtmdAccUtopiaConfig Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>507</b>
Data Fields.....	507
Detailed Description.....	508
Field Documentation.....	508

# Table of Contents

<b>IxAtmdAccUtopiaConfig::UtRxConfig_ Struct Reference [IXP425 ATM Driver Access</b>	
<b>(IxAtmdAcc) Utopia Control API].....</b>	<b>512</b>
Data Fields.....	512
Detailed Description.....	513
Field Documentation.....	513
 <b>IxAtmdAccUtopiaConfig::UtRxDefineIdle_ Struct Reference [IXP425 ATM Driver Access</b>	
<b>(IxAtmdAcc) Utopia Control API].....</b>	<b>518</b>
Data Fields.....	518
Detailed Description.....	518
Field Documentation.....	519
 <b>IxAtmdAccUtopiaConfig::UtRxEnableFields_ Struct Reference [IXP425 ATM Driver Access</b>	
<b>(IxAtmdAcc) Utopia Control API].....</b>	<b>520</b>
Data Fields.....	520
Detailed Description.....	522
Field Documentation.....	522
 <b>IxAtmdAccUtopiaConfig::UtRxStatsConfig_ Struct Reference [IXP425 ATM Driver Access</b>	
<b>(IxAtmdAcc) Utopia Control API].....</b>	<b>528</b>
Data Fields.....	528
Detailed Description.....	528
Field Documentation.....	529
 <b>IxAtmdAccUtopiaConfig::UtRxTransTable0_ Struct Reference [IXP425 ATM Driver Access</b>	
<b>(IxAtmdAcc) Utopia Control API].....</b>	<b>530</b>
Data Fields.....	530
Detailed Description.....	530
Field Documentation.....	530
 <b>IxAtmdAccUtopiaConfig::UtRxTransTable1_ Struct Reference [IXP425 ATM Driver Access</b>	
<b>(IxAtmdAcc) Utopia Control API].....</b>	<b>533</b>
Data Fields.....	533
Detailed Description.....	533
Field Documentation.....	533
 <b>IxAtmdAccUtopiaConfig::UtRxTransTable2_ Struct Reference [IXP425 ATM Driver Access</b>	
<b>(IxAtmdAcc) Utopia Control API].....</b>	<b>536</b>
Data Fields.....	536
Detailed Description.....	536
Field Documentation.....	536
 <b>IxAtmdAccUtopiaConfig::UtRxTransTable3_ Struct Reference [IXP425 ATM Driver Access</b>	
<b>(IxAtmdAcc) Utopia Control API].....</b>	<b>539</b>
Data Fields.....	539
Detailed Description.....	539
Field Documentation.....	539

# Table of Contents

<b>IxAtmdAccUtopiaConfig::UtRxTransTable4_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>542</b>
Data Fields.....	542
Detailed Description.....	542
Field Documentation.....	542
<b>IxAtmdAccUtopiaConfig::UtRxTransTable5_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>545</b>
Data Fields.....	545
Detailed Description.....	545
Field Documentation.....	545
<b>IxAtmdAccUtopiaConfig::UtSysConfig_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>546</b>
Data Fields.....	546
Detailed Description.....	547
Field Documentation.....	547
<b>IxAtmdAccUtopiaConfig::UtTxConfig_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>549</b>
Data Fields.....	549
Detailed Description.....	550
Field Documentation.....	550
<b>IxAtmdAccUtopiaConfig::UtTxDefineIdle_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>554</b>
Data Fields.....	554
Detailed Description.....	554
Field Documentation.....	555
<b>IxAtmdAccUtopiaConfig::UtTxEnableFields_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>556</b>
Data Fields.....	556
Detailed Description.....	557
Field Documentation.....	557
<b>IxAtmdAccUtopiaConfig::UtTxStatsConfig_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>561</b>
Data Fields.....	561
Detailed Description.....	561
Field Documentation.....	562
<b>IxAtmdAccUtopiaConfig::UtTxTransTable0_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>563</b>
Data Fields.....	563
Detailed Description.....	563
Field Documentation.....	563

# Table of Contents

<b>IxAtmdAccUtopiaConfig::UtTxTransTable1_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>566</b>
Data Fields.....	566
Detailed Description.....	566
Field Documentation.....	566
<b>IxAtmdAccUtopiaConfig::UtTxTransTable2_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>569</b>
Data Fields.....	569
Detailed Description.....	569
Field Documentation.....	569
<b>IxAtmdAccUtopiaConfig::UtTxTransTable3_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>572</b>
Data Fields.....	572
Detailed Description.....	572
Field Documentation.....	572
<b>IxAtmdAccUtopiaConfig::UtTxTransTable4_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>575</b>
Data Fields.....	575
Detailed Description.....	575
Field Documentation.....	575
<b>IxAtmdAccUtopiaConfig::UtTxTransTable5_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>578</b>
Data Fields.....	578
Detailed Description.....	578
Field Documentation.....	578
<b>IxAtmdAccUtopiaStatus Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>579</b>
Data Fields.....	579
Detailed Description.....	579
Field Documentation.....	580
<b>IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>582</b>
Data Fields.....	582
Detailed Description.....	583
Field Documentation.....	583
<b>IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_ Struct Reference [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API].....</b>	<b>585</b>
Data Fields.....	585
Detailed Description.....	585
Field Documentation.....	585

# Table of Contents

<b>IxAtmmPortCfg Struct Reference [IXP425 ATM Manager (IxAtmm) API].....</b>	<b>588</b>
Data Fields.....	588
Detailed Description.....	588
Field Documentation.....	588
<b>IxAtmmVc Struct Reference [IXP425 ATM Manager (IxAtmm) API].....</b>	<b>590</b>
Data Fields.....	590
Detailed Description.....	590
Field Documentation.....	590
<b>IxAtmScheduleTable Struct Reference [IXP425 ATM Types (IxAtmTypes)].....</b>	<b>592</b>
Data Fields.....	592
Detailed Description.....	592
Field Documentation.....	592
<b>IxAtmScheduleTableEntry Struct Reference [IXP425 ATM Types (IxAtmTypes)].....</b>	<b>594</b>
Data Fields.....	594
Detailed Description.....	594
Field Documentation.....	594
<b>IxAtmTrafficDescriptor Struct Reference [IXP425 ATM Types (IxAtmTypes)].....</b>	<b>596</b>
Data Fields.....	596
Detailed Description.....	596
Field Documentation.....	597
<b>IxEthAccMacAddr Struct Reference [IXP425 Ethernet Access (IxEthAcc) API].....</b>	<b>599</b>
Data Fields.....	599
Detailed Description.....	599
Field Documentation.....	599
<b>IxEthDBMacAddr Struct Reference [IXP425 Ethernet Database (IxEthDB) API].....</b>	<b>600</b>
Data Fields.....	600
Detailed Description.....	600
<b>IxEthDBPortDefinition Struct Reference [IXP425 Ethernet Database Port Definitions (IxEthDBPortDefs)].....</b>	<b>601</b>
Data Fields.....	601
Detailed Description.....	601
<b>IxEthEthObjStats Struct Reference [IXP425 Ethernet Access (IxEthAcc) API].....</b>	<b>602</b>
Data Fields.....	602
Detailed Description.....	603
Field Documentation.....	603
<b>IxHssAccCodeletStats Struct Reference.....</b>	<b>606</b>
Data Fields.....	606
Detailed Description.....	606

# Table of Contents

<b>IxHssAccConfigParams Struct Reference [IXP425 HSS Access (IxHssAcc) API].....</b>	<b>607</b>
Data Fields.....	607
Detailed Description.....	607
Field Documentation.....	607
<b>IxHssAccHdlcMode Struct Reference [IXP425 HSS Access (IxHssAcc) API].....</b>	<b>610</b>
Data Fields.....	610
Detailed Description.....	610
Field Documentation.....	610
<b>IxHssAccPktHdlcFraming Struct Reference [IXP425 HSS Access (IxHssAcc) API].....</b>	<b>612</b>
Data Fields.....	612
Detailed Description.....	612
Field Documentation.....	612
<b>IxHssAccPortConfig Struct Reference [IXP425 HSS Access (IxHssAcc) API].....</b>	<b>614</b>
Data Fields.....	614
Detailed Description.....	615
Field Documentation.....	615
<b>IxMbufPool Struct Reference [IXP425 OS Memory Buffer Pool Management (IxOsBuffPoolMgt) API].....</b>	<b>619</b>
Data Fields.....	619
Detailed Description.....	619
Field Documentation.....	620
<b>IxNpeA_NpePacketDescriptor Struct Reference [IXP425 NPE-A (IxNpeA) API].....</b>	<b>622</b>
Data Fields.....	622
Detailed Description.....	622
Field Documentation.....	623
<b>IxNpeA_RxAtmVc Struct Reference [IXP425 NPE-A (IxNpeA) API].....</b>	<b>625</b>
Data Fields.....	625
Detailed Description.....	626
Field Documentation.....	626
<b>IxNpeA_TxAtmVc Struct Reference [IXP425 NPE-A (IxNpeA) API].....</b>	<b>628</b>
Data Fields.....	628
Detailed Description.....	628
Field Documentation.....	628
<b>IxNpeDllImageId Struct Reference [IXP425 NPE-Downloader (IxNpeDI) API].....</b>	<b>631</b>
Data Fields.....	631
Detailed Description.....	631
Field Documentation.....	631



# Table of Contents

<b>IxNpeMhMessage Struct Reference [IXP425 NPE Message Handler (IxNpeMh) API].....</b>	<b>633</b>
Data Fields.....	633
Detailed Description.....	633
Field Documentation.....	633
<b>IxOamITU610Cell Struct Reference.....</b>	<b>634</b>
Data Fields.....	634
Detailed Description.....	634
<b>IxOamITU610GenericPayload Struct Reference.....</b>	<b>635</b>
Data Fields.....	635
Detailed Description.....	635
<b>IxOamITU610LbPayload Struct Reference.....</b>	<b>636</b>
Data Fields.....	636
Detailed Description.....	636
<b>IxOamITU610Payload Union Reference.....</b>	<b>637</b>
Data Fields.....	637
Detailed Description.....	637
<b>IxPerfProfAccBusPmuResults Struct Reference [IXP425 Performance Profiling (IxPerfProfAcc) API].....</b>	<b>638</b>
Data Fields.....	638
Detailed Description.....	638
Field Documentation.....	638
<b>IxPerfProfAccXcycleResults Struct Reference [IXP425 Performance Profiling (IxPerfProfAcc) API].....</b>	<b>639</b>
Data Fields.....	639
Detailed Description.....	639
Field Documentation.....	639
<b>IxPerfProfAccXscalePmuEvtCnt Struct Reference [IXP425 Performance Profiling (IxPerfProfAcc) API].....</b>	<b>641</b>
Data Fields.....	641
Detailed Description.....	641
Field Documentation.....	641
<b>IxPerfProfAccXscalePmuResults Struct Reference [IXP425 Performance Profiling (IxPerfProfAcc) API].....</b>	<b>642</b>
Data Fields.....	642
Detailed Description.....	642
Field Documentation.....	643
<b>IxPerfProfAccXscalePmuSamplePcProfile Struct Reference [IXP425 Performance Profiling (IxPerfProfAcc) API].....</b>	<b>645</b>
Data Fields.....	645

# Table of Contents

<b>IxPerfProfAccXscalePmuSamplePcProfile Struct Reference [IXP425 Performance Profiling (IxPerfProfAcc) API]</b>	
Detailed Description.....	645
Field Documentation.....	645
<b>IxQMgrQInlinedReadWriteInfo Struct Reference [IXP425 Queue Manager (IxQMgr) API].....</b>	<b>646</b>
Data Fields.....	646
Detailed Description.....	646
Field Documentation.....	646
<b>ixUARTDev Struct Reference [IXP425 UART Access (IxUARTAcc) API, IXP425 UART Access (IxUARTAcc) API].....</b>	<b>649</b>
Data Fields.....	649
Detailed Description.....	649
Field Documentation.....	649
<b>ixUARTStats Struct Reference [IXP425 UART Access (IxUARTAcc) API, IXP425 UART Access (IxUARTAcc) API].....</b>	<b>651</b>
Data Fields.....	651
Detailed Description.....	651
<b>PacketisedStats Struct Reference.....</b>	<b>652</b>
Data Fields.....	652
Detailed Description.....	652
<b>USBDevice Struct Reference [IXP425 USB Driver Public API].....</b>	<b>653</b>
Data Fields.....	653
Detailed Description.....	653
Field Documentation.....	653
<b>USBDeviceCounters Struct Reference.....</b>	<b>655</b>
Data Fields.....	655
Detailed Description.....	655
<b>USBSetupPacket Struct Reference [IXP425 USB Driver Public API].....</b>	<b>657</b>
Data Fields.....	657
Detailed Description.....	657

# IXP425 Operating System Services Library (IxOSSL) API

This service provides a layer of OS dependency services.

## Data Structures

struct **ix\_ossl\_thread\_main\_info\_t**  
*This type defines thread main info.*

struct **ix\_ossl\_time\_t**  
*This type defines OSSL time.*

## Defines

#define **IX\_OSSL\_ERROR\_SUCCESS**  
*This symbol defines an error token that indicates the successful completion of the OSSL calls.*

#define **IX\_OSSL\_ERROR\_FAILURE**  
*This symbol defines an error token that indicates the failed completion of the OSSL calls.*

#define **IX\_OSSL\_WAIT\_FOREVER**  
*This symbol is useful for specifying an 'indefinite wait' timeout value in ix\_ossl\_sem\_take and ix\_ossl\_mutex\_lock function calls.*

#define **IX\_OSSL\_WAIT\_NONE**  
*This symbol is useful for specifying a 'no wait' timeout value in ix\_ossl\_sem\_take and ix\_ossl\_mutex\_lock function calls.*

#define **BILLION**  
*Define a constant for the value 1 billion, used by OSSL TIME macros. Equivalent to (1000 million nanoseconds / second).*

#define **TICKS\_PER\_NSEC**  
*The number of OS Ticks per nano-second on Linux.*

#define **IX\_OSSL\_TIME\_EQ(a, b)**  
*Compares a operand with b operand. Returns true if they are equal and false otherwise.*

#define **IX\_OSSL\_TIME\_GT(a, b)**  
*Compares a operand with b operand. Returns true if a > b, and false otherwise.*

#define **IX\_OSSL\_TIME\_LT(a, b)**

*Compares a operand with b operand. Returns true if  $a < b$ , and false otherwise.*

**#define IX\_OSSL\_TIME\_ISZERO(a)**

*This macro checks if the operand a is zero. Returns true if a is zero (both sec and nsec fields must be zero) and false otherwise.*

**#define IX\_OSSL\_TIME\_SET(a, b)**

*This macro sets operand a to the value of operand b.*

**#define IX\_OSSL\_TIME\_ADD(a, b)**

*This macro performs  $a += b$  operation.*

**#define IX\_OSSL\_TIME\_SUB(a, b)**

*This macro performs  $a -= b$  operation.*

**#define IX\_OSSL\_TIME\_NORMALIZE(a)**

*This macro normalizes the value of a. If 'a.nsec' >  $10^9$ , it is decremented by  $10^9$  and 'a.sec' is incremented by 1.*

**#define IX\_OSSL\_TIME\_VALID(a)**

*This macro checks whether a is a valid **ix\_ossl\_time\_t** i.e.  $0 \leq a.nsec < 10^9$ . Returns true if a is valid and false otherwise.*

**#define IX\_OSSL\_TIME\_ZERO(a)**

*This macro sets the value of a to zero.*

**#define IX\_OSSL\_TIME\_CONVERT\_TO\_TICK(a, b)**

*This macro converts b value in **ix\_ossl\_time\_t** to a value in os ticks.*

## Typedefs

**typedef os\_thread\_t ix\_ossl\_thread\_t**

*This type defines OSSL thread type.*

**typedef os\_sem\_t ix\_ossl\_sem\_t**

*This type defines OSSL semaphore type.*

**typedef os\_mutex\_t ix\_ossl\_mutex\_t**

*This type defines OSSL mutex type.*

**typedef ix\_error\_t (\*ix\_ossl\_thread\_entry\_point\_t)(void \*arg, void \*\*ptrRetObj)**

*This function type defines OSSL thread entry point function. .*

**typedef unsigned int ix\_ossl\_size\_t**

*This type describes a generic size type.*

## Enumerations

```
enum ix_ossll_error_code {  
    IX_OSSL_ERROR_SUCCESS,  
    IX_OSSL_ERROR_INVALID_ARGUMENTS,  
    IX_OSSL_ERROR_INVALID_OPERATION,  
    IX_OSSL_ERROR_THREAD_CALL_FAILURE,  
    IX_OSSL_ERROR_INVALID_PID,  
    IX_OSSL_ERROR_INVALID_TID,  
    IX_OSSL_ERROR_OS_CALL_FAILURE,  
    IX_OSSL_ERROR_TIMEOUT,  
    IX_OSSL_ERROR_NOMEM,  
    IX_OSSL_ERROR_NOSYS  
}
```

*This type defines error codes returned by OSSL calls.*

```
enum ix_ossll_sem_state {  
    IX_OSSL_SEM_UNAVAILABLE,  
    IX_OSSL_SEM_AVAILABLE  
}
```

*This type defines OSSL binary semaphore states.*

```
enum ix_ossll_mutex_state {  
    IX_OSSL_MUTEX_UNLOCK,  
    IX_OSSL_MUTEX_LOCK  
}
```

*This type defines OSSL mutex states.*

```
enum ix_ossll_thread_priority {  
    IX_OSSL_THREAD_PRI_HIGH,  
    IX_OSSL_THREAD_PRI_MEDIUM,  
    IX_OSSL_THREAD_PRI_LOW  
}
```

*This type define OSSL thread priority levels.*

## Functions

IX\_EXPORT\_FUNCTION **ix\_ossll\_thread\_create** (ix\_ossll\_thread\_entry\_point\_t entryPoint, void \*arg,  
ix\_error **ix\_ossll\_thread\_t** \*ptrTid)  
*create a thread*

IX\_EXPORT\_FUNCTION  
ix\_error **ix\_ossll\_thread\_get\_id** (ix\_ossll\_thread\_t \*ptrTid)  
*get id of calling thread*

IX\_EXPORT\_FUNCTION  
void \* **ix\_ossll\_thread\_main\_wrapper** (void \*ptrThreadInfo)  
*wrapper for user-provided thread function*  
  
**ix\_ossll\_thread\_exit** (ix\_error retError, void \*retObj)

IX\_EXPORT\_FUNCTION

void

*Causes the calling thread to exit.*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_thread\_kill** (ix\_ossl\_thread\_t tid)

*kills the specified thread*

IX\_EXPORT\_FUNCTION **ix\_ossl\_thread\_set\_priority** (ix\_ossl\_thread\_t tid, ix\_ossl\_thread\_priority

ix\_error priority)

*sets the priority of the specified thread*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_thread\_delay** (int ticks)

*delay the current task for specified number of OS ticks*

IX\_EXPORT\_FUNCTION **Nos\_thread\_create** (void \*(\*start\_routine)(void \*),

int **ix\_ossl\_thread\_main\_info\_t** \*ptrThreadInfo, **ix\_ossl\_thread\_t** \*ptrTid,

os\_error \*osError)

*creates a thread (use **ix\_ossl\_thread\_create** instead)*

IX\_EXPORT\_FUNCTION

int **os\_thread\_get\_id** (ix\_ossl\_thread\_t \*ptrTid)

*get thread id (use **ix\_ossl\_thread\_get\_id** instead)*

IX\_EXPORT\_FUNCTION

int **os\_thread\_exit** (void \*ptrRetObj)

*exits the calling thread (use **ix\_ossl\_thread\_exit** instead)*

IX\_EXPORT\_FUNCTION

int **os\_thread\_kill** (ix\_ossl\_thread\_t tid, os\_error \*osError)

*kill the specified thread (use **ix\_ossl\_thread\_kill** instead)*

IX\_EXPORT\_FUNCTION **Nos\_thread\_set\_priority** (ix\_ossl\_thread\_t \*tid, ix\_ossl\_thread\_priority

ix\_error priority, os\_error \*osError)

*sets priority of a thread (use **ix\_ossl\_thread\_set\_priority** instead)*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_tick\_get** (int \*pticks)

*gets number of OS ticks per second*

IX\_EXPORT\_FUNCTION

int **os\_sleep** (ix\_uint32 sleeptime\_ms, os\_error \*osError)

*causes the calling thread to sleep for specified time (milliseconds)*

IX\_EXPORT\_FUNCTION

int **os\_sleep\_tick** (ix\_uint32 sleeptime\_ticks, os\_error \*osError)

*causes the calling thread to sleep for specified time (os ticks)*

IX\_EXPORT\_FUNCTION

int **os\_time\_get** (ix\_ossl\_time\_t \*ptime, os\_error \*osError)

*returns the system time with nano-second resolution*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_sem\_init** (int start\_value, **ix\_ossl\_sem\_t** \*sid)  
*initialises a new semaphore*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_sem\_take** (**ix\_ossl\_sem\_t** sid, ix\_uint32 timeout)  
*take a semaphore, and block if semaphore not available*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_sem\_give** (**ix\_ossl\_sem\_t** sid)  
*give back a semaphore*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_sem\_flush** (**ix\_ossl\_sem\_t** sid, int \*result)  
*unblocks all threads pending on the semaphore*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_sem\_fini** (**ix\_ossl\_sem\_t** sid)  
*terminate the semaphore (free semaphore resources)*

IX\_EXPORT\_FUNCTION

int **os\_thread\_sema\_create** (int value, **ix\_ossl\_sem\_t** \*sid, os\_error \*osError)  
*create a thread semaphore*

IX\_EXPORT\_FUNCTION

int **os\_thread\_sema\_P** (**ix\_ossl\_sem\_t** \*sid, ix\_uint32 timeout, os\_error \*osError)  
*pend on a semaphore (with timeout)*

IX\_EXPORT\_FUNCTION

int **os\_thread\_sema\_V** (**ix\_ossl\_sem\_t** \*sid, os\_error \*osError)  
*release a semaphore*

IX\_EXPORT\_FUNCTION

int **os\_thread\_sema\_destroy** (**ix\_ossl\_sem\_t** \*sid, os\_error \*osError)  
*destroy semaphore object*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_mutex\_init** (**ix\_ossl\_mutex\_state** start\_state, **ix\_ossl\_mutex\_t** \*mid)  
*initialises a new mutex object*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_mutex\_lock** (**ix\_ossl\_mutex\_t** mid, ix\_uint32 timeout)  
*lock a mutex*

IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_mutex\_unlock** (**ix\_ossl\_mutex\_t** mid)  
*unlocks a mutex*

**ix\_ossl\_mutex\_fini** (**ix\_ossl\_mutex\_t** mid)

IX\_EXPORT\_FUNCTION

ix\_error  
*free a mutex*

IX\_EXPORT\_FUNCTION **ix\_ossl\_mutex\_create** (**ix\_ossl\_mutex\_state** start\_state, **ix\_ossl\_mutex\_t**  
int \*mid, os\_error \*osError)  
*creates a thread mutex object*

IX\_EXPORT\_FUNCTION **ix\_ossl\_mutex\_lock** (**ix\_ossl\_mutex\_t** \*mutex, ix\_uint32 timeout, os\_error  
int \*osError)  
*lock a mutex*

IX\_EXPORT\_FUNCTION  
int **ix\_ossl\_mutex\_unlock** (**ix\_ossl\_mutex\_t** \*mutex, os\_error \*osError)  
*unlock a mutex*

IX\_EXPORT\_FUNCTION  
int **ix\_ossl\_mutex\_destroy** (**ix\_ossl\_mutex\_t** \*mutex, os\_error \*osError)  
*destroy a mutex object*

ix\_error **ix\_ossl\_sleep** (ix\_uint32 sleeptime\_ms)  
*causes calling thread to sleep for specified time (milliseconds)*

IX\_EXPORT\_FUNCTION  
ix\_error **ix\_ossl\_sleep\_tick** (ix\_uint32 sleeptime\_ticks)  
*causes calling thread to sleep for specified time (os ticks)*

IX\_EXPORT\_FUNCTION  
ix\_error **ix\_ossl\_time\_get** (**ix\_ossl\_time\_t** \*ptime)  
*gets current system time with nano-second resolution*

IX\_EXPORT\_FUNCTION  
void \* **ix\_ossl\_malloc** (**ix\_ossl\_size\_t** arg\_Size)  
*allocate a block of memory*

IX\_EXPORT\_FUNCTION  
void **ix\_ossl\_free** (void \*arg\_pMemory)  
*free a block of memory*

IX\_EXPORT\_FUNCTION **ix\_ossl\_memcpy** (void \*arg\_pDest, const void \*arg\_pSrc, **ix\_ossl\_size\_t**  
void \* arg\_Count)  
*copy number of bytes from one memory location to another*

IX\_EXPORT\_FUNCTION  
void \* **ix\_ossl\_memset** (void \*arg\_pDest, int arg\_pChar, **ix\_ossl\_size\_t** arg\_Count)  
*fill a region of memory with a specified byte value*

IX\_EXPORT\_FUNCTION  
ix\_error **ix\_ossl\_message\_log\_init** (void)  
*initialise the error message logging facility*



IX\_EXPORT\_FUNCTION

ix\_error **ix\_ossl\_message\_log** (char \*arg\_pFmtString,...)  
*log a printf-style message*

---

## Detailed Description

This service provides a layer of OS dependency services.

This file contains the prototypes of OS-independent wrapper functions which allow the programmer not to be tied to a specific operating system. The OSSL functions can be divided into three classes:

1) synchronization-related wrapper functions around thread system calls 2) thread-related wrapper functions around thread calls 3) transactor/workbench osapi calls — defined in osApi.h

Both 1 and 2 classes of functions provide Thread Management, Thread Synchronization, Mutual Exclusion and Timer primitives. Namely, creation and deletion functions as well as the standard "wait" and "exit". Additionally, a couple of utility functions which enable to pause the execution of a thread are also provided.

The 3rd class provides a slew of other OSAPI functions to handle Transactor/WorkBench OS calls.

**Note:**

WHEREVER POSSIBLE, PLEASE USE THE EQUIVALENT API FUNCTION FROM THE  
**IXP425 OS Services (IxOsServices) API COMPONENT** INSTEAD IF ONE EXISTS.

---

## Define Documentation

```
#define BILLION
```

Define a constant for the value 1 billion, used by OSSL TIME macros. Equivalent to (1000 million nanoseconds / second).

Definition at line **394** of file **ix\_ossl.h**.

```
#define IX_OSSL_ERROR_FAILURE
```

This symbol defines an error token that indicates the failed completion of the OSSL calls.

Definition at line **351** of file **ix\_ossl.h**.

```
#define IX_OSSL_ERROR_SUCCESS
```

This symbol defines an error token that indicates the successful completion of the OSSL calls.

Definition at line **342** of file **ix\_oss.h**.

```
#define IX_OSSL_TIME_ADD ( a,  
                           b )
```

This macro performs a += b operation.

**Parameters:**

- a* **ix\_oss\_time\_t** (in|out) – operand a
- b* **ix\_oss\_time\_t** (in) – operand b

Definition at line **485** of file **ix\_oss.h**.

```
#define IX_OSSL_TIME_CONVERT_TO_TICK ( a,  
                                       b )
```

This macro converts b value in **ix\_oss\_time\_t** to a value in os ticks.

**Parameters:**

- a* unsigned int (out) – operand a
- b* **ix\_oss\_time\_t** (in) – operand b

Definition at line **570** of file **ix\_oss.h**.

```
#define IX_OSSL_TIME_EQ ( a,  
                          b )
```

Compares a operand with b operand. Returns true if they are equal and false otherwise.

**Parameters:**

- a* **ix\_oss\_time\_t** (in) – operand a
- b* **ix\_oss\_time\_t** (in) – operand b

**Returns:**

true if a == b and false otherwise.

Definition at line **416** of file **ix\_oss.h**.

```
#define IX_OSSL_TIME_GT ( a,  
                          b )
```

Compares a operand with b operand. Returns true if a > b, and false otherwise.

**Parameters:**

- a* **ix\_ossl\_time\_t** (in) – operand a
- b* **ix\_ossl\_time\_t** (in) – operand b

**Returns:**

true if  $a > b$  and false otherwise.

Definition at line **431** of file **ix\_ossl.h**.

```
#define IX_OSSL_TIME_ISZERO ( a )
```

This macro checks if the operand a is zero. Returns true if a is zero (both sec and nsec fields must be zero) and false otherwise.

**Parameters:**

- a* **ix\_ossl\_time\_t** (in) – operand a

**Returns:**

true if a is zero and false otherwise.

Definition at line **461** of file **ix\_ossl.h**.

```
#define IX_OSSL_TIME_LT ( a,  
                        b )
```

Compares a operand with b operand. Returns true if  $a < b$ , and false otherwise.

**Parameters:**

- a* **ix\_ossl\_time\_t** (in) – operand a
- b* **ix\_ossl\_time\_t** (in) – operand b

**Returns:**

true if  $a < b$  and false otherwise.

Definition at line **446** of file **ix\_ossl.h**.

```
#define IX_OSSL_TIME_NORMALIZE ( a )
```

This macro normalizes the value of a. If 'a.nsec' >  $10^9$ , it is decremented by  $10^9$  and 'a.sec' is incremented by 1.

**Parameters:**

- a* **ix\_ossl\_time\_t** (in|out) – operand a

Definition at line **523** of file **ix\_ossl.h**.

```
#define IX_OSSL_TIME_SET ( a,  
                           b )
```

This macro sets operand a to the value of operand b.

**Parameters:**

- a* **ix\_ossl\_time\_t** (out) – operand a
- b* **ix\_ossl\_time\_t** (in) – operand b

Definition at line **473** of file **ix\_ossl.h**.

```
#define IX_OSSL_TIME_SUB ( a,  
                           b )
```

This macro performs a -= b operation.

**Parameters:**

- a* **ix\_ossl\_time\_t** (in|out) – operand a
- b* **ix\_ossl\_time\_t** (in) – operand b

Definition at line **502** of file **ix\_ossl.h**.

```
#define IX_OSSL_TIME_VALID ( a )
```

This macro checks whether a is a valid **ix\_ossl\_time\_t** i.e.  $0 \leq a.\text{nsec} < 10^9$ . Returns true if a is valid and false otherwise.

**Parameters:**

- a* **ix\_ossl\_time\_t** (in) – operand a

**Returns:**

true if a is valid **ix\_ossl\_time\_t** and false otherwise

Definition at line **541** of file **ix\_ossl.h**.

```
#define IX_OSSL_TIME_ZERO ( a )
```

This macro sets the value of a to zero.

**Parameters:**

- a* **ix\_ossl\_time\_t** (in|out) – operand a

Definition at line **553** of file **ix\_ossl.h**.

```
#define IX_OSSL_WAIT_FOREVER
```

This symbol is useful for specifying an 'indefinite wait' timeout value in `ix_ossl_sem_take` and `ix_ossl_mutex_lock` function calls.

Definition at line **361** of file **ix\_ossl.h**.

```
#define IX_OSSL_WAIT_NONE
```

This symbol is useful for specifying a 'no wait' timeout value in `ix_ossl_sem_take` and `ix_ossl_mutex_lock` function calls.

Definition at line **370** of file **ix\_ossl.h**.

```
#define TICKS_PER_NSEC
```

The number of OS Ticks per nano-second on Linux.

Definition at line **402** of file **ix\_ossl.h**.

---

## Typedef Documentation

```
ix_ossl_mutex_t
```

This type defines OSSL mutex type.

Definition at line **215** of file **ix\_ossl.h**.

```
ix_ossl_sem_t
```

This type defines OSSL semaphore type.

Definition at line **207** of file **ix\_ossl.h**.

```
ix_ossl_size_t
```

This type describes a generic size type.

Definition at line **1255** of file **ix\_ossl.h**.

```
ix_ossl_thread_entry_point_t
```

This function type defines OSSL thread entry point function. .

**Parameters:**

*void*        \* arg (in) – pointer to a custom thread argument  
*void*        \*\* ptrRetObj (out) – address where a pointer to a custom data structure or object will be returned by the thread on exit.

Definition at line **247** of file **ix\_ossl.h**.

ix\_ossl\_thread\_t

This type defines OSSL thread type.

Definition at line **199** of file **ix\_ossl.h**.

---

## Enumeration Type Documentation

enum ix\_ossl\_error\_code

This type defines error codes returned by OSSL calls.

**Enumeration values:**

<i>IX_OSSL_ERROR_SUCCESS</i>	success
<i>IX_OSSL_ERROR_INVALID_ARGUMENTS</i>	invalid arguments
<i>IX_OSSL_ERROR_INVALID_OPERATION</i>	invalid operation
<i>IX_OSSL_ERROR_THREAD_CALL_FAILURE</i>	thread operation failed
<i>IX_OSSL_ERROR_INVALID_PID</i>	invalid process id
<i>IX_OSSL_ERROR_INVALID_TID</i>	invalid thread id
<i>IX_OSSL_ERROR_OS_CALL_FAILURE</i>	invalid arguments
<i>IX_OSSL_ERROR_TIMEOUT</i>	OS operation failed.
<i>IX_OSSL_ERROR_NOMEM</i>	memory unavailable
<i>IX_OSSL_ERROR_NOSYS</i>	system resource unavailable

Definition at line **277** of file **ix\_ossl.h**.

enum ix\_ossl\_mutex\_state

This type defines OSSL mutex states.

**Enumeration values:**

<i>IX_OSSL_MUTEX_UNLOCK</i>	Mutex unlocked.
<i>IX_OSSL_MUTEX_LOCK</i>	Mutex locked.

Definition at line **312** of file **ix\_ossl.h**.

enum ix\_ossl\_sem\_state

This type defines OSSL binary semaphore states.

**Enumeration values:**

<i>IX_OSSL_SEM_UNAVAILABLE</i>	Semaphore unavailable.
<i>IX_OSSL_SEM_AVAILABLE</i>	Semaphore available.

Definition at line **298** of file **ix\_ossl.h**.

enum ix\_ossl\_thread\_priority

This type define OSSL thread priority levels.

**Enumeration values:**

<i>IX_OSSL_THREAD_PRI_HIGH</i>	High priority thread.
<i>IX_OSSL_THREAD_PRI_MEDIUM</i>	Medium priority thread.
<i>IX_OSSL_THREAD_PRI_LOW</i>	Low priority thread.

Definition at line **325** of file **ix\_ossl.h**.

---

## Function Documentation

IX\_EXPORT\_FUNCTION void ix\_ossl\_free ( void \* *arg\_pMemory* )

free a block of memory

This function will free a memory block specified by the passed address. The ix\_ossl\_free function

deallocates a memory block (*arg\_pMemory*) that was previously allocated by a call to *ix\_ossl\_malloc*. If *arg\_pMemory* is NULL, the pointer is ignored and *ix\_ossl\_free* immediately returns. Attempting to free an invalid pointer (a pointer to a memory block that was not allocated by *ix\_ossl\_malloc*) may affect subsequent allocation requests and cause errors.

See ***ixOsServCacheDmaFree*** for an alternative.

**Parameters:**

*arg\_pMemory* void\* (in) – address of the memory block to be deallocated.

```
IX_EXPORT_FUNCTION void* ix_ossl_malloc ( ix_ossl_size_t arg_Size )
```

allocate a block of memory

This function will allocate a memory block. the function returns a void pointer to the allocated space, or NULL if there is insufficient memory available. To return a pointer to a type other than void, use a type cast on the return value. The storage space pointed to by the return value is guaranteed to be suitably aligned for storage of any type of object. If size is 0, *ix\_ossl\_malloc* allocates a zero-length item in the heap and returns a valid pointer to that item. Always check the return from *ix\_ossl\_malloc*, even if the amount of memory requested is small.

See ***ixOsServCacheDmaAlloc*** for an alternative.

**Parameters:**

*arg\_Size* ix\_ossl\_size\_t (in) – the size of the memory block requested.

**Returns:**

Returns a valid address if successful or a NULL for failure.

```
IX_EXPORT_FUNCTION void* ix_ossl_memcpy ( void *      arg_pDest,  
                                          const void * arg_pSrc,  
                                          ix_ossl_size_t arg_Count  
                                          )
```

copy number of bytes from one memory location to another

This function will copy memory bytes between buffers. The *ix\_ossl\_memcpy* function copies count bytes of *arg\_pSrc* to *arg\_pDest*. If the source and destination overlap, this function does not ensure that the original source bytes in the overlapping region are copied before being overwritten.

**Parameters:**

*arg\_pDest* void\* (in|out) – destination buffer address

*arg\_pSrc* const void\* (in) – source buffer address

*arg\_Count* ix\_ossl\_size\_t – number of bytes to copy

**Returns:**

Returns the address of the destination buffer.



```
IX_EXPORT_FUNCTION void* ix_ossl_memset ( void *      arg_pDest,
                                         int          arg_pChar,
                                         ix_ossl_size_t arg_Count
                                         )
```

fill a region of memory with a specified byte value

This function sets buffers to a specified character. The ix\_ossl\_memset function sets the first arg\_Count bytes of arg\_pDest to the character arg\_Char.

**Parameters:**

*arg\_pDest* void\* (in/out) – destination buffer address  
*arg\_pChar* int (in) – character to set  
*arg\_Count* ix\_ossl\_size\_t (in) – number of characters to set

**Returns:**

Returns the address of the destination buffer.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_message_log ( char * arg_pFmtString,
                                                ...
                                                )
```

log a printf-style message

This routine is used to log a specified message. This routine's syntax is similar to printf() – a format string is followed by a variable number of arguments which will be interpreted and formatted according to the fmt\_string passed as the first argument. Further details will be provided on where the messages will be logged!

See **ixOsServLog** for an alternative.

**Parameters:**

*arg\_pFmtString* char\* (in) – format string for the log message

**Returns:**

IX\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_message_log_init ( void )
```

initialise the error message logging facility

This function is used to initialize the error message logging. For each OS the messages will be logged into an implementation dependent stream. Further details will be provided on where the messages will be logged! This function should be called before any call to **ix\_ossl\_message\_log()**.

See **ixOsServLogLevelSet** for an alternative.

**Returns:**

IX\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_mutex_fini ( ix_ossl_mutex_t mid )
```

free a mutex

This function frees a mutex. 'mid' is the id mutex id. The mutex is deleted, all resources are freed. Any threads pending on this mutex will be unblocked and return an error.

**Parameters:**

*mid* ix\_ossl\_mutex\_t (in) – mutex id

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_mutex_init ( ix_ossl_mutex_state start_state,  
                                                ix_ossl_mutex_t * mid  
                                                )
```

initialises a new mutex object

This function initializes a new mutex. 'mid' is a pointer to an ix\_ossl\_mutex\_t. Upon success, '\*mid' will contain the mutex id. 'start\_state' is the initial locking state.

**Note:**

**Parameters:**

*start\_state* ix\_ossl\_mutex\_state (in) – 'start\_state' is initial locking state. Valid values are :–  
'OSSL\_MUTEX\_LOCK': Lock the mutex 'OSSL\_MUTEX\_UNLOCK': Leave the mutex unlocked  
*mid* ix\_ossl\_mutex\_t\* (out) – pointer to id of the mutex created

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_mutex_lock ( ix_ossl_mutex_t mid,  
                                                ix_uint32 timeout  
                                                )
```

lock a mutex

This function locks the mutex. If the mutex is already locked, the thread will block. If the time indicated in 'timeout' is reached, the thread will unblock and return error indication. If the timeout is set to 'IX\_OSSL\_WAIT\_NONE', the thread will never block (this is trylock). If the timeout is set to 'IX\_OSSL\_WAIT\_FOREVER', the thread will block until the mutex is unlocked.

**Parameters:**

*mid* ix\_ossl\_mutex\_t (in) – mutex id.  
*timeout* ix\_uint32 (in) – timeout value of type ix\_uint32 expressed in milliseconds.

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_mutex_unlock ( ix_ossl_mutex_t mid )
```

unlocks a mutex

This function unlocks the mutex. 'mid' is the mutex id. If there are threads pending on the mutex, the next one is given the lock. If there are no pending threads, then the mutex is unlocked.

**Parameters:**

*mid* ix\_ossl\_mutex\_t (in) – mutex id

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_sem_fini ( ix_ossl_sem_t sid )
```

terminate the semaphore (free semaphore resources)

This function frees a semaphore. 'sid' is semaphore id. The semaphore is terminated, all resources are freed. The threads pending on this semaphore will be released and return an error.

**Parameters:**

*sid* ix\_ossl\_sem\_t (in) – semaphore id

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_sem_flush ( ix_ossl_sem_t sid,  
                                                int *      result  
                                                )
```

unblocks all threads pending on the semaphore

This function unblocks all pending threads without altering the semaphore count. 'sid' is the id for the semaphore. '\*result' will be non-zero if a thread was unblocked during this call.

**Parameters:**

*sid* ix\_ossl\_sem\_t (in) – semaphore id

*result* int\* (out) – the value referred will be non-zero if a thread was unblocked during this call

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_sem_give ( ix_ossl_sem_t sid )
```

give back a semaphore

This function causes the next available thread in the pend queue to be unblocked. If no thread is pending on this semaphore, the semaphore becomes 'full'.

**Parameters:**

*sid* **ix\_ossl\_sem\_t** (in) – semaphore id.

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_sem_init ( int start_value,  
                                              ix_ossl_sem_t * sid  
                                              )
```

initialises a new semaphore

This function initializes a new semaphore. 'sid' is a pointer to an **ix\_ossl\_sem\_t**. Upon success, '\*sid' will be the semaphore id used in all other ix\_ossl\_sem functions. The newly created semaphore will be initialized the value of 'start\_value'.

**Parameters:**

*start\_value* int (in) – initial value of the semaphore

*sid* **ix\_ossl\_sem\_t**\* (out) – Address where the newly created semaphore id will returned.

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_sem_take ( ix_ossl_sem_t sid,  
                                              ix_uint32 timeout  
                                              )
```

take a semaphore, and block if semaphore not available

If the semaphore is 'empty', the calling thread is blocked. If the semaphore is 'full', it is taken and control is returned to the caller. If the time indicated in 'timeout' is reached, the thread will unblock and return an error indication. If the timeout is set to 'IX\_OSSL\_WAIT\_NONE', the thread will never block; if it is set to 'IX\_OSSL\_WAIT\_FOREVER', the thread will block until the semaphore is available.

**Parameters:**

*sid* **ix\_ossl\_sem\_t** (in) – semaphore id.

*timeout* ix\_uint32 (in) – timeout value of type ix\_uint32 expressed in milliseconds

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
ix_error ix_ossleep ( ix_uint32 sleep_time_ms )
```

causes calling thread to sleep for specified time (milliseconds)

This function causes the calling thread to sleep for the specified time.

**Parameters:**

*sleep\_time\_ms* ix\_uint32 (in) – sleep time specified in milliseconds.

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

: In VxWorks this function has a resolution dictated by sysClkRateGet(). Very small delays could be truncated to 0 i.e. no delay at all. See **ixOsServTaskSleep()** for an alternative.

```
IX_EXPORT_FUNCTION ix_error ix_ossleep_tick ( ix_uint32 sleep_time_ticks )
```

causes calling thread to sleep for specified time (os ticks)

This function causes the calling thread to sleep for the time specified in OS ticks.

**Parameters:**

*sleep\_time\_ticks* ix\_uint32 (in) – sleep time specified in os ticks.

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossleep_thread_create ( ix_ossleep_thread_entry_point_t entryPoint,  
                                                       void * arg,  
                                                       ix_ossleep_thread_t * ptrTid  
                                                       )
```

create a thread

This function creates a cancellable thread that will execute the user-provided entry point function. Custom arguments can be passed to this function using the "arg" argument.

**Parameters:**

*entryPoint* ix\_ossleep\_thread\_entry\_point\_t (in) – thread's entry point function.

*arg* void\* (in) – pointer to custom arguments that will be passed to entry point function as the first argument.

*ptrTid* ix\_ossleep\_thread\_t\* (out) – address at which the thread id of the newly created thread will be returned

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_thread_delay ( int ticks )
```

delay the current task for specified number of OS ticks

This function causes the current task to delay for the specified number of OS ticks. Control of the CPU is relinquished during this time allowing other system tasks a chance to execute.

**Parameters:**

*ticks* int (in) – number of OS ticks to delay task.

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or IX\_OSSL\_ERROR\_FAILURE for failure.

```
IX_EXPORT_FUNCTION void ix_ossl_thread_exit ( ix_error retError,  
                                              void * retObj  
                                              )
```

Causes the calling thread to exit.

This function causes the calling thread to exit. It gives the opportunity to the caller to pass back to a waiting parent a pointer to a custom data structure and an ix\_error token.

**Parameters:**

*retError* ix\_error (in) – ix\_error token to be returned to the waiting parent (0 if no error is returned)

*retObj* void\* (in) – pointer to custom data structure returned to the waiting parent on thread exit. It is used for post-mortem debugging. (null if no data structure is returned)

**Returns:**

none

```
IX_EXPORT_FUNCTION ix_error ix_ossl_thread_get_id ( ix_ossl_thread_t * ptrTid )
```

get id of calling thread

This function returns id of the calling thread.

**Parameters:**

*ptrTid* ix\_ossl\_thread\_t\* (out) – address at which the id of the calling thread will be returned

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_thread_kill ( ix_ossl_thread_t tid )
```

kills the specified thread

Kills the running thread specified by its thread id. Because the thread will be killed instantly, the caller must be extremely careful when using this function as the thread will not have time to release any of the resources it is currently owning. `ix_ossl_thread_exit` should be used to delete a thread and its resources instead!.

**Parameters:**

*tid* `ix_ossl_thread_t` (in) – id of the thread to be killed

**Returns:**

`IX_OSSL_ERROR_SUCCESS` if successful or a valid `ix_error` token for failure.

**Warning:**

This function does not kill linux kernel threads. It only sends the `SIG_TERM` signal and it is the responsibility of the thread to check for this signal (see `signal_pending`) and terminate itself.

```
IX_EXPORT_FUNCTION void* ix_ossl_thread_main_wrapper ( void * ptrThreadInfo )
```

wrapper for user-provided thread function

This function provides the needed pthread-compliant entry point function. It basically acts as a wrapper around the user-provided function. It still does one important thing which is to set the cancellation type of the thread to `ASYNCHRONOUS` (which translates into instantaneous")

**Parameters:**

*ptrThreadInfo* `void*` (in) – pointer to our temporary structure containing pointers to the thread's entry point function and its argument structure

**Returns:**

`void *`

```
IX_EXPORT_FUNCTION ix_error ix_ossl_thread_set_priority ( ix_ossl_thread_t      tid,  
                                                         ix_ossl_thread_priority priority  
                                                         )
```

sets the priority of the specified thread

This function sets the priority of the indicated thread. Possible values for 'priority' are `IX_OSSL_THREAD_PRI_HIGH`, `IX_OSSL_THREAD_PRI_MED`, and `IX_OSSL_THREAD_PRI_LOW`.

**Parameters:**

*tid* `ix_ossl_thread_t` (in) – id of the thread

*priority* `ix_ossl_thread_priority` (in) – valid priority values are: `IX_OSSL_THREAD_PRI_HIGH`, `IX_OSSL_THREAD_PRI_MED`, and `IX_OSSL_THREAD_PRI_LOW`.

**Returns:**

`IX_OSSL_ERROR_SUCCESS` if successful or a valid `ix_error` token for failure.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_tick_get ( int * pticks )
```

gets number of OS ticks per second

This function returns the number of os ticks per second.

**Parameters:**

*pticks* int\* (out) – pointer to location where data will be returned.

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS.

```
IX_EXPORT_FUNCTION ix_error ix_ossl_time_get ( ix_ossl_time_t * ptime )
```

gets current system time with nano-second resolution

This function places the current value of a timer, in seconds and nano-seconds, into an **ix\_ossl\_time\_t** structure pointed by 'ptime'. This function does not provide a time-of-day. The intention is to provide a nano-second resolution time.

**Parameters:**

*ptime* ix\_ossl\_time\_t\* (out) – pointer to 'ix\_ossl\_time\_t' structure where data will be returned.

**Returns:**

IX\_OSSL\_ERROR\_SUCCESS if successful or a valid ix\_error token for failure.

```
IX_EXPORT_FUNCTION int os_sleep ( ix_uint32 sleeptime_ms,  
                                os_error * osError  
                                )
```

causes the calling thread to sleep for specified time (milliseconds)

This function causes the calling thread to sleep for the specified time.

**Parameters:**

<i>sleeptime_ms</i>	ix_uint32 (in) – sleep time specified in milliseconds.
<i>osError</i>	os_error* (out) – pointer to the datastructure where the error conditions are returned.

```
IX_EXPORT_FUNCTION int os_sleep_tick ( ix_uint32 sleeptime_ticks,  
                                       os_error * osError  
                                       )
```

causes the calling thread to sleep for specified time (os ticks)



This function causes the calling thread to sleep for the time specified in OS ticks.

**Parameters:**

*sleep\_time\_ticks* ix\_uint32 (in) – sleep time specified in OS ticks.  
*osError* os\_error\* (out) – pointer to the datastructure where the error conditions are returned.

```
IX_EXPORT_FUNCTION int os_thread_create ( void *(*start_routine)(void *),  
                                          ix_ossl_thread_main_info_t * ptrThreadInfo,  
                                          ix_ossl_thread_t * ptrTid,  
                                          os_error * osError  
                                          )
```

creates a thread (use **ix\_ossl\_thread\_create** instead)

Create a cancellable thread that will execute the user–provided entry point function. Custom arguments can be passed to this function using the "arg" argument.

**Parameters:**

*start\_routine* void \* (\*)(void \*) (in ) – pointer to thread's entry point function  
*ptrThreadInfo* ix\_ossl\_thread\_main\_info\_t\* (in) – pointer to custom argument structure that will be passed to entry point function  
*ptrTid* ix\_ossl\_thread\_t\* (out) – address at which the thread id of the newly created thread will be returned  
*osError* os\_error\* (out) – pointer to the datastructure where OS error codes are returned.

```
IX_EXPORT_FUNCTION int os_thread_exit ( void * ptrRetObj )
```

exits the calling thread (use **ix\_ossl\_thread\_exit** instead)

This function causes the calling thread to exit. It gives the opportunity (this is not a requirement!) to the caller to pass back to a waiting parent a pointer to a custom data structure and an ix\_error token.

**Parameters:**

*ptrRetObj* void\* (out) – pointer to custom data structure (null if no data structure is returned)

```
IX_EXPORT_FUNCTION int os_thread_get_id ( ix_ossl_thread_t * ptrTid )
```

get thread id (use **ix\_ossl\_thread\_get\_id** instead)

Returns the thread id of the calling thread

**Parameters:**

*ptrTid* ix\_ossl\_thread\_t\* (out) – address at which the thread id of the inquiring thread will be returned

```

IX_EXPORT_FUNCTION int os_thread_kill ( ix_ossl_thread_t tid,
                                         os_error * osError
                                         )

```

kill the specified thread (use **ix\_ossl\_thread\_kill** instead)

Kills the running thread specified by its thread id.

**Parameters:**

*tid*        **ix\_ossl\_thread\_t** (in) – id of the thread to be killed  
*osError* **os\_error\*** (out) – pointer to the datastructure where OS error conditions are returned.

```

IX_EXPORT_FUNCTION int os_thread_mutex_create ( ix_ossl_mutex_state start_state,
                                                ix_ossl_mutex_t * mid,
                                                os_error * osError
                                                )

```

creates a thread mutex object

Create a thread mutex object.

See **ixOsServMutexInit** for an alternative.

**Parameters:**

*start\_state* **ix\_ossl\_mutex\_state** (in) – the value that the mutex state should be initialized to.  
*mid*        **ix\_ossl\_mutex\_t\*** (out) – id of the thread mutex created.  
*osError*    **os\_error\*** (out) – pointer to the datastructure where the error conditions are returned.

```

IX_EXPORT_FUNCTION int os_thread_mutex_destroy ( ix_ossl_mutex_t * mutex,
                                                os_error * osError
                                                )

```

destroy a mutex object

Destroy the thread mutex object.

see **ixOsServMutexDestroy** for an alternative

**Parameters:**

*mutex*     **ix\_ossl\_mutex\_t\*** (in) – pointer to the thread mutex object.  
*osError*    **os\_error\*** (out) – pointer to the datastructure where the error conditions are returned.

```

IX_EXPORT_FUNCTION int os_thread_mutex_lock ( ix_ossl_mutex_t * mutex,
                                              ix_uint32      timeout,
                                              os_error *      osError
                                              )

```

lock a mutex

This function locks the mutex. If the mutex is already locked, the task will block. If the time indicated in 'timeout' is reached, the task will unblock and return error indication. If timeout is set to '0', the task will never block.

See **ixOsServMutexLock** for an alternative.

**Parameters:**

*mutex*    **ix\_ossl\_mutex\_t**\* (in) – pointer to the thread mutex object.  
*timeout*   **ix\_uint32** (in) – 'timeout' value  
*osError*   **os\_error**\* (out) – pointer to the datastructure where the error conditions are returned.

```

IX_EXPORT_FUNCTION int os_thread_mutex_unlock ( ix_ossl_mutex_t * mutex,
                                                os_error *      osError
                                                )

```

unlock a mutex

This function unlocks the mutex. If there are tasks pending on the mutex, the next one is given the lock. If there are no pending tasks, then the mutex is unlocked.

See **ixOsServMutexUnlock** for an alternative.

**Parameters:**

*mutex*    **ix\_ossl\_mutex\_t**\* (in) – pointer to the thread mutex object.  
*osError*   **os\_error**\* (out) – pointer to the datastructure where the error conditions are returned.

```

IX_EXPORT_FUNCTION int os_thread_sema_create ( int      value,
                                              ix_ossl_sem_t * sid,
                                              os_error *      osError
                                              )

```

create a thread semaphore

Create a thread semaphore object.

**Parameters:**

*value*    int (in) – value that the semaphore should be initialize to.  
*sid*      **ix\_ossl\_sem\_t**\* (out) – id of the thread semaphore created.  
*osError*   **os\_error**\* (out) – pointer to the datastructure where the error conditions are returned.

```
IX_EXPORT_FUNCTION int os_thread_sema_destroy ( ix_ossl_sem_t * sid,
                                              os_error * osError
                                              )
```

destroy semaphore object

Destroy the thread semaphore object.

**Parameters:**

*sid*        **ix\_ossl\_sem\_t**\* (in) – pointer to the thread semaphore object.  
*osError* **os\_error**\* (out) – pointer to the datastructure where the error conditions are returned.

```
IX_EXPORT_FUNCTION int os_thread_sema_P ( ix_ossl_sem_t * sid,
                                          ix_uint32        timeout,
                                          os_error *        osError
                                          )
```

pend on a semaphore (with timeout)

Waits on the specified semaphore until the units are available or timeout occurs.

**Parameters:**

*sid*        **ix\_ossl\_sem\_t**\* (in) – pointer to the semaphore object.  
*timeout* **ix\_uint32** (in) – timeout value.  
*osError* **os\_error**\* (out) – pointer to the datastructure where the error conditions are returned.

```
IX_EXPORT_FUNCTION int os_thread_sema_V ( ix_ossl_sem_t * sid,
                                          os_error *        osError
                                          )
```

release a semaphore

This function releases the specified semaphore and the semaphore state becomes available after this function call.

**Parameters:**

*sid*        **ix\_ossl\_sem\_t**\* (in) – pointer to the thread semaphore object.  
*osError* **os\_error**\* (out) – pointer to the datastructure where the error conditions are returned.

```
IX_EXPORT_FUNCTION ix_error os_thread_set_priority ( ix_ossl_thread_t *        tid,
                                                    ix_ossl_thread_priority priority,
                                                    os_error *        osError
                                                    )
```

sets priority of a thread (use **ix\_ossl\_thread\_set\_priority** instead)

This function sets the priority of the indicated thread. possible values for 'priority' are IX\_OSSL\_PRI\_HIGH, IX\_OSSL\_PRI\_MED, and IX\_OSSL\_PRI\_LOW. The effect of priority is OS dependant.

**Parameters:**

*tid* ix\_ossl\_thread\_t\* (in) – pointer to the thread object  
*priority* ix\_ossl\_thread\_priority (in) – priority level.  
*osError* os\_error\* (out) – pointer to the datastructure where OS error conditions are returned.

```
IX_EXPORT_FUNCTION int os_time_get ( ix_ossl_time_t * ptime,  
                                     os_error * osError  
                                     )
```

returns the system time with nano-second resolution

This function places the current value of time, in (seconds, nanoseconds), into the '\*ptime' structure. This function does not provide a time-of-day. The purpose is to provide a nano-second resolution time.

**Parameters:**

*ptime* ix\_ossl\_time\_t\* (out) – address to the time structure.  
*osError* os\_error\* (out) – pointer to the datastructure where the error conditions are returned.

# IXP425 ADSL Driver API

The public API for the IXP425 ADSL Driver.

## Typedefs

```
typedef void(* IxAdslStateChangeCallback )(UINT32 lineNum, IxAdslLineState lineState)
```

*Callback function to indicate of the changes on the line state.*

## Enumerations

```
enum IxAdslStatus {  
    IX_ADSL_STATUS_SUCCESS,  
    IX_ADSL_STATUS_FAIL,  
    IX_ADSL_STATUS_UNSUPPORTED_MODE,  
    IX_ADSL_STATUS_ALREADY_DOWN  
}
```

*These status will be used by the APIs to return to the client.*

```
enum IxAdslLineState {  
    IX_ADSL_LINE_STATE_UP_DUAL_LATENCY,  
    IX_ADSL_LINE_STATE_WAIT_FOR_ACTIVATING,  
    IX_ADSL_LINE_STATE_ACTIVATING,  
    IX_ADSL_LINE_STATE_DOWN,  
    IX_ADSL_LINE_STATE_UP_FASTCHANNEL,  
    IX_ADSL_LINE_STATE_UP_INTERLEAVECHANNEL,  
    IX_ADSL_LINE_STATE_INVALID  
}
```

*These status will be used to indicate the line state.*

```
enum IxAdslLineType {  
    IX_ADSL_AUTOSELECT,  
    IX_ADSL_GLITE,  
    IX_ADSL_DMT,  
    IX_ADSL_ANSI,  
    IX_ADSL_LOOPBACK,  
    IX_ADSL_INVALID_MODE  
}
```

*Used to indicate the type of ADSL line type used.*

```
enum IxAdslPhyType {  
    IX_ADSL_PHY_CPE,  
    IX_ADSL_PHY_INVALID  
}
```

*Used to indicate the ADSL physical type – CPE.*

## Functions

**PUBLIC** **ixAdslLineOpen** (UINT32 lineNum, **ixAdslLineType** lineType, **ixAdslPhyType** phyType)  
**ixAdslStatus**

*Open the given ADSL line in the specified mode and type.*

**PUBLIC**

**ixAdslStatus** **ixAdslLineClose** (UINT32 lineNum)

*Closes a previously opened ADSL line.*

**PUBLIC** **ixAdslLineStateChangeCallbackRegister** (UINT32 lineNum,  
**ixAdslStatus** **ixAdslStateChangeCallback** lineChangeCallbackFn)

*This is a notification registration procedure that gets called if the line state of the given ADSL line changes. The maximum callbacks that can be registered is defined as IX\_ADSL\_SIZEOF\_CALLBACK\_LIST (The default value is 10).*

**PUBLIC**

**ixAdslLineState** **ixAdslLineStateGet** (UINT32 lineNum)

*Returns the current state of the given ADSL line.*

**PUBLIC** **UINT32** **ixAdslLineRateUpstreamGet** (UINT32 lineNum)

*Return the current upstream line speed of the given ADSL line.*

**PUBLIC** **UINT32** **ixAdslLineRateDownstreamGet** (UINT32 lineNum)

*Return the current downstream line speed of the given ADSL line.*

**PUBLIC**

**ixAdslStatus** **ixAdslDyingGaspEnable** (UINT32 lineNum)

*Enables the function that informs ATU-C when condition that leads to shutdown of the given Adsl line has been detected.*

**PUBLIC** **ixAdslVendorCodeSet** (UINT32 lineNum, **UINT8** ixAdslItuVendoridCountrycode,  
**ixAdslStatus** **UINT8** ixAdslItuVendoridVendorcode1, **UINT8** ixAdslItuVendoridVendorcode2, **UINT8** ixAdslItuVendoridVendorcode3, **UINT8** ixAdslItuVendoridVendorcode4, **UINT8** ixAdslItuVendoridVendorspecific1, **UINT8** ixAdslItuVendoridVendorspecific2)

*Set the vendor specific bytes in the given ADSL line.*

**PUBLIC** **void** **ixAdslShow** (UINT32 lineNum)

*This function will show the current statistics associated with the given ADSL Line.*

**PUBLIC** **void** **ixAdslMemoryUnmap** (**void**)

*This function will unmap the dynamically allocated addresses.*

---

## Detailed Description

The public API for the IXP425 ADSL Driver.

---

# Typedef Documentation

```
typedef void(* IxAdslStateChangeCallback)( UINT32 lineNum, IxAdslLineState lineState)
```

Callback function to indicate of the changes on the line state.

Definition at line **178** of file **IxAdsl.h**.

---

## Enumeration Type Documentation

```
enum IxAdslLineState
```

These status will be used to indicate the line state.

### *Enumeration values:*

<i>IX_ADSL_LINE_STATE_UP_DUAL_LATENCY</i>	The line is in showtime state.
<i>IX_ADSL_LINE_STATE_WAIT_FOR_ACTIVATING</i>	Fast & Interleaved Channel The line is waiting for the peer to activate.
<i>IX_ADSL_LINE_STATE_ACTIVATING</i>	The line is negotiating with its peer.
<i>IX_ADSL_LINE_STATE_DOWN</i>	The line is down.
<i>IX_ADSL_LINE_STATE_UP_FASTCHANNEL</i>	The line is in showtime state.  Fast Channel
<i>IX_ADSL_LINE_STATE_UP_INTERLEAVECHANNEL</i>	The line is in showtime state.  Interleaved Channel
<i>IX_ADSL_LINE_STATE_INVALID</i>	ADSL line in an unknown state.

Definition at line **91** of file **IxAdsl.h**.

```
enum IxAdslLineType
```

Used to indicate the type of ADSL line type used.

### *Enumeration values:*

<i>IX_ADSL_AUTOSELECT</i>	This is an auto-select mode for the CPE to auto-configure based on the CO/DSLAM line type; DMT, ANSI or G.lite.
<i>IX_ADSL_GLITE</i>	G.lite line type.
<i>IX_ADSL_DMT</i>	DMT line type.
<i>IX_ADSL_ANSI</i>	ANSI line type.
<i>IX_ADSL_LOOPBACK</i>	Utopia Loopback line type.



*IX\_ADSL\_INVALID\_MODE* Used internally to indicate last valid enum.

Definition at line **132** of file **IxAdsl.h**.

enum IxAdslPhyType

Used to indicate the ADSL physical type – CPE.

**Note:**

IxAdslPhyType is declared as an enum due to forward compatibility to support CO (fast and interleaved mode) in the future.

**Enumeration values:**

*IX\_ADSL\_PHY\_CPE* Adsl type is CPE.

*IX\_ADSL\_PHY\_INVALID* Adsl type is invalid.

Definition at line **161** of file **IxAdsl.h**.

enum IxAdslStatus

These status will be used by the APIs to return to the client.

**Enumeration values:**

*IX\_ADSL\_STATUS\_SUCCESS* Successful API execution.

*IX\_ADSL\_STATUS\_FAIL* Failed API execution.

*IX\_ADSL\_STATUS\_UNSUPPORTED\_MODE* Unsupported mode type for IxAdslLineOpen function.

*IX\_ADSL\_STATUS\_ALREADY\_DOWN* Line is already down.

Definition at line **70** of file **IxAdsl.h**.

---

## Function Documentation

ixAdslDyingGaspEnable ( UINT32 *lineNum* )

Enables the function that informs ATU–C when condition that leads to shutdown of the given Adsl line has been detected.

**Note:**

The parameter lineNum exists for future Multi–PHY support. Only lineNum = 0 is valid.

Blocking : This call is a non–blocking

Impacts On Global Data: None

Pre–Conditions: Task level calls only. Only Available in ATU–R.

Post-Conditions: None.

Exceptions: None.

**Parameters:**

*lineNum* [in] is the parameter showing which ADSL line is being used.

**Returns:**

◇ IX\_ADSL\_STATUS\_SUCCESS – Dying Gasp is enabled. successfully.

◇ IX\_ADSL\_STATUS\_FAILED – Failed to enable Dying Gasp.

```
ixAdslLineClose ( UINT32 lineNum )
```

Closes a previously opened ADSL line.

The line will closed and put in the idle state.

**Note:**

The parameter *lineNum* exists for future Multi-PHY support. Only *lineNum* = 0 is valid.

Blocking : Non-blocking

Impacts On Global Data: Sets the *lineEnable* State . Notifies a callback routine.

Pre-Conditions: Code should only be called from task level.

Post-Conditions: No cleanup after this call is required.

**Parameters:**

*lineNum* [in] is the parameter showing which ADSL line is being used and to be closed.

**Returns:**

◇ IX\_ADSL\_STATUS\_SUCCESS – Line was closed successfully.

◇ IX\_ADSL\_STATUS\_FAILED – Line failed to close properly.

◇ IX\_ADSL\_STATUS\_ALREADY\_DOWN – Line was not open before close.

```
ixAdslLineOpen ( UINT32 lineNum,  
                  IxAdslLineType lineType,  
                  IxAdslPhyType phyType  
                  )
```

Open the given ADSL line in the specified mode and type.

Opens the given ADSL line in the specified mode and puts it in the 'Showtime' state, i.e. available to carry user data.

**Note:**

- The parameter *lineNum* exists for future Multi-PHY support. Only *lineNum* = 0 is valid.
- The parameter *phyType* exists for future CO support.

Blocking : This call may block for several seconds while the link is established.

Impacts On Global Data: Sets the *lineEnable* State .

Pre-Conditions: The code must only be called once the operating system is running, i.e. do not call as part of the hardware init as this code requires base services such as *Atmd*, *Atmm*, *Atm* scheduler and *Utopia*. The code should only be called from task level.

Post-Conditions: No cleanup after this call is required.

Exceptions: None.

**Parameters:**

*lineNum* [in] is the parameter showing which ADSL line is being used.

*lineType* [in] indicates type of ADSL to be opened.

*phyType* [in] is the type of Phy used – CPE.

**Returns:**

◇ IX\_ADSL\_STATUS\_SUCCESS – Line was opened and is in 'showtime' state

◇ IX\_ADSL\_STATUS\_FAILED – Line failed to open properly.

◇ IX\_ADSL\_STATUS\_UNSUPPORTED\_MODE – Illegal ADSL type for *IxAdslLineOpen* function.

```
ixAdslLineRateDownstreamGet ( UINT32 lineNum )
```

Return the current downstream line speed of the given ADSL line.

The data returned by this API is the received (Rx) rate of the line the ATU device.

**Note:**

The parameter *lineNum* exists for future Multi-PHY support. Only *lineNum* = 0 is valid.

Blocking : This call is a non-blocking

Impacts On Global Data: None

Pre-Conditions: Task level calls only.

Post-Conditions: None.

Exceptions: None.

**Parameters:**

*lineNum* [in] is the parameter showing which ADSL line is being used.

**Returns:**

◇ Integer – bit rate in kbits/second. N.B. Returns zero if the line is not in 'Showtime' state.

```
ixAdslLineRateUpstreamGet ( UINT32 lineNum )
```

Return the current upstream line speed of the given ADSL line.

**Note:**

The parameter *lineNum* exists for future Multi-PHY support. Only *lineNum* = 0 is valid.

The data returned by this API represents the transmit (Tx) rate of the line from the ATU device.

Blocking : This call is a non-blocking

Impacts On Global Data: None

Pre-Conditions: Task level calls only.

Post-Conditions: None.

Exceptions: None.

**Parameters:**

*lineNum* [in] is the parameter showing which ADSL line is being used.

**Returns:**

◇ Integer – bit rate in kbits/second. N.B. Returns zero if the line is not in 'Showtime' state.

```
ixAdslLineStateChangeCallbackRegister ( UINT32 lineNum,  
                                         IxAdslStateChangeCallback lineChangeCallbackFn  
                                         )
```

This is a notification registration procedure that gets called if the line state of the given ADSL line changes. The maximum callbacks that can be registered is defined as `IX_ADSL_SIZEOF_CALLBACK_LIST` (The default value is 10).

**Note:**

The parameter *lineNum* exists for future Multi-PHY support. Only *lineNum* = 0 is valid.

Blocking : This call is a non-blocking function.

Impacts On Global Data: This sets a global callback handler.

Pre-Conditions: There are no pre conditions to this call.

Post-Conditions: A global line state handler for ADSL line state changes shall be registered. It is advisable to register the callback before the given ADSL line is opened.

Exceptions: None.

**Parameters:**

*lineNum* [in] is the parameter showing which ADSL line is being used.  
*lineChangeCallbackFn* [in] is the callback function that will be invoked when there is a state change

**Returns:**

- ◇ IX\_ADSL\_STATUS\_SUCCESS – The callback function was registered successfully.
- ◇ IX\_ADSL\_STATUS\_FAILED – Internal error, registration of the callback function failed.

```
ixAdslLineStateGet ( UINT32 lineNum )
```

Returns the current state of the given ADSL line.

**Note:**

The parameter lineNum exists for future Multi-PHY support. Only lineNum = 0 is valid.

Blocking : This call is a non-blocking

Impacts On Global Data: None

Pre-Conditions: Task level calls only.

Post-Conditions: None.

Exceptions: None.

**Parameters:**

*lineNum* [in] is the parameter showing which ADSL line is being used.

**Returns:**

- ◇ IX\_ADSL\_LINE\_STATE\_UP\_FAST – The line is in show time state.
- ◇ IX\_ADSL\_LINE\_STATE\_WAIT\_FOR\_ACTIVATING – The line is waiting fo the peer to activate.
- ◇ IX\_ADSL\_LINE\_STATE\_ACTIVATING – The line is negotiating with its peer.
- ◇ IX\_ADSL\_LINE\_STATE\_DOWN – The line is idle.
- ◇ IX\_ADSL\_LINE\_STATE\_INVALID – The line is in an unknown state.

## ixAdslMemoryUnmap ( void )

This function will unmap the dynamically allocated addresses.

Blocking : This call is non-blocking.

Impacts On Global Data: None

Pre-Conditions: Task level calls only.

Post-Conditions: None.

Exceptions: None.

### **Parameters:**

*None*

### **Returns:**

*None*

## ixAdslShow ( UINT32 lineNum )

This function will show the current statistics associated with the given ADSL Line.

### **Note:**

The parameter lineNum exists for future Multi-PHY support. Only lineNum = 0 is valid.

The list of statistics to be shown by IxAdslShow:

- Controller SW Version
- ADSL Line State
- Line Number
- Upstream and Downstream Rate
- Training Statistics for Upstream and Downstream Rates
- ADSL Near End Operational Data such as Upstream Relative Capacity Occupancy, Noise Margin Upstream, Output Pwr Downstream, Attenuation Upstream, Downstream Fast Bitrate, Downstream Interleaved Bitrate, Near-end defect bitmap, Loss of Frame (secs), Loss of Cell delineation (secs), Loss of Signal (secs), Loss of Margin (secs), Errored seconds, HEC and FEC Errors.
- ADSL Far End Operational Data such as Downstream Relative Capacity Occupancy, Noise Margin Downstream, Output Pwr Upstream, Attenuation Downstream, Upstream Fast Bitrate, Upstream Interleaved Bitrate, Far-end defect bitmap
- Tx and Rx ATM Cell Counters

Blocking : This call is non-blocking.

Impacts On Global Data: None

Pre-Conditions: Task level calls only.

Post-Conditions: None.

Exceptions: None.

**Parameters:**

*lineNum* [in] is the parameter showing which ADSL line is being used.

**Returns:**

None

```
ixAdslVendorCodeSet ( UINT32 lineNum,  
                      UINT8  ixAdslItuVendoridCountrycode,  
                      UINT8  ixAdslItuVendoridVendorcode1,  
                      UINT8  ixAdslItuVendoridVendorcode2,  
                      UINT8  ixAdslItuVendoridVendorcode3,  
                      UINT8  ixAdslItuVendoridVendorcode4,  
                      UINT8  ixAdslItuVendoridVendorspecific1,  
                      UINT8  ixAdslItuVendoridVendorspecific2  
                      )
```

Set the vendor specific bytes in the given ADSL line.

The vendor ID must be set before the line is open, if not the default vendor ID shall be sent to the peer modem upon request.

**Note:**

- Vendor specific values are taken by the phy and linked together to form a single code
- Note that the parameter lineNum exists for future Multi-PHY support. Only lineNum = 0 is valid.

Blocking : This call is a non-blocking

Impacts On Global Data: Sets an internal ADSL global data structure. This shall be used for all subsequent ADSL line open commands.

Pre-Conditions: Task level calls only.

Post-Conditions: None.

Exceptions: None.

**Parameters:**

<i>lineNum</i>	[in] is the parameter showing which ADSL line is being used.
<i>ixAdslItuVendoridCountrycode</i>	[in] is the vendor country code that are predefined in standards.
<i>ixAdslItuVendoridVendorcode1</i>	[in] is the vendor code 1 that are predefined in standards.
<i>ixAdslItuVendoridVendorcode2</i>	[in] is the vendor code 2 that are predefined in standards.
<i>ixAdslItuVendoridVendorcode3</i>	[in] is the vendor code 3 that are predefined in standards.

*ixAdslItuVendoridVendorcode4* [in] is the vendor code 4 that are predefined in standards.  
*ixAdslItuVendoridVendorspecific1* [in] is the vendor specific 1 that are predefined in standards.  
*ixAdslItuVendoridVendorspecific2* [in] is the vendor specific 2 that are predefined in standards.

**Returns:**

- ◇ IX\_ADSL\_STATUS\_SUCCESS – Set Vendor Code successful.
- ◇ IX\_ADSL\_STATUS\_FAILED – Set Vendor Code failed because the line is up.



# IXP425 Assertion Macros (IxAssert) API

Assertion support.

## Defines

`#define IX_ASSERT(c)`

*Assert macro, assert the condition is true. This will not be compiled out. N.B. will result in a system crash if it is false.*

`#define IX_ENSURE(c, str)`

*Ensure macro, ensure the condition is true. This will be conditionally compiled out and may be used for unit/integration test purposes.*

---

## Detailed Description

Assertion support.

---

## Define Documentation

```
#define IX_ASSERT ( c )
```

Assert macro, assert the condition is true. This will not be compiled out. N.B. will result in a system crash if it is false.

Definition at line **76** of file **IxAssert.h**.

```
#define IX_ENSURE ( c,  
                  str )
```

Ensure macro, ensure the condition is true. This will be conditionally compiled out and may be used for unit/integration test purposes.

Definition at line **84** of file **IxAssert.h**.

# IXP425 ATM Driver Access (IxAtmdAcc) API

The public API for the IXP425 Atm Driver Data component.

## Defines

**#define IX\_ATMDACC\_WARNING**

*Warning return code.*

**#define IX\_ATMDACC\_BUSY**

*Busy return code.*

**#define IX\_ATMDACC\_RESOURCES\_STILL\_ALLOCATED**

*Disconnect return code.*

**#define IX\_ATMDACC\_DEFAULT\_REPLENISH\_COUNT**

*Default resources usage for RxVcFree replenish mechanism.*

**#define IX\_ATMDACC\_OAM\_TX\_VPI**

*The reserved value used for the dedicated OAM Tx connection. This "well known" value is used by atmdAcc and its clients to discriminate the OAM channel, and should be chosen so that it does not coincide with the VPI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VPI will fail.*

**#define IX\_ATMDACC\_OAM\_TX\_VCI**

*The reserved value used for the dedicated OAM Tx connection. This "well known" value is used by atmdAcc and its clients to discriminate the OAM channel, and should be chosen so that it does not coincide with the VCI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VCI will fail.*

**#define IX\_ATMDACC\_OAM\_RX\_PORT**

*The reserved dummy PORT used for all dedicated OAM Rx connections. Note that this is not a real port but must have a value that lies within the valid range of port values.*

**#define IX\_ATMDACC\_OAM\_RX\_VPI**

*The reserved value value used for the dedicated OAM Rx connection. This value should be chosen so that it does not coincide with the VPI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VPI will fail.*

**#define IX\_ATMDACC\_OAM\_RX\_VCI**

*The reserved value value used for the dedicated OAM Rx connection. This value should be chosen so that it does not coincide with the VCI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VCI will fail.*

## Typedefs

```
typedef
unsigned int IxAtmdAccUserId
    User-supplied Id.

typedef void(* IxAtmdAccRxVcRxCallback )(IxAtmLogicalPort port, IxAtmdAccUserId userId,
IxAtmdAccPduStatus status, IxAtmdAccClpStatus clp, IX_MBUF *mbufPtr)
    Rx callback prototype.

typedef void(* IxAtmdAccRxVcFreeLowCallback )(IxAtmdAccUserId userId)
    Callback prototype for free buffer level is low.

typedef void(* IxAtmdAccTxVcBufferReturnCallback )(IxAtmdAccUserId userId, IX_MBUF *mbufPtr)
    Buffer callback prototype.
```

## Enumerations

```
enum IxAtmdAccPduStatus {
    IX_ATMDACC_AAL0_VALID,
    IX_ATMDACC_OAM_VALID,
    IX_ATMDACC_AAL2_VALID,
    IX_ATMDACC_AAL5_VALID,
    IX_ATMDACC_AAL5_PARTIAL,
    IX_ATMDACC_AAL5_CRC_ERROR,
    IX_ATMDACC_MBUF_RETURN
}
IxAtmdAcc Pdu status .:

enum IxAtmdAccAalType {
    IX_ATMDACC_AAL5,
    IX_ATMDACC_AAL2,
    IX_ATMDACC_AAL0_48,
    IX_ATMDACC_AAL0_52,
    IX_ATMDACC_OAM,
    IX_ATMDACC_MAX_SERVICE_TYPE
}
IxAtmdAcc AAL Service Type .:

enum IxAtmdAccClpStatus {
    IX_ATMDACC_CLP_NOT_SET,
    IX_ATMDACC_CLP_SET
}
IxAtmdAcc CLP indication.
```

# Functions

**PUBLIC**  
**IX\_STATUS ixAtmdAccInit** (void)  
*Initialise the IxAtmdAcc Component.*

**PUBLIC void ixAtmdAccShow** (void)  
*Show IxAtmdAcc configuration on a per port basis.*

**PUBLIC void ixAtmdAccStatsShow** (void)  
*Show all IxAtmdAcc stats.*

**PUBLIC void ixAtmdAccStatsReset** (void)  
*Reset all IxAtmdAcc stats.*

**PUBLIC ixAtmdAccRxVcConnect** (IxAtmLogicalPort port, unsigned int vpi, unsigned int vci,  
**IX\_STATUS IxAtmdAccAalType** aalServiceType, **IxAtmRxQueueId** rxQueueId, **IxAtmdAccUserId**  
userCallbackId, **IxAtmdAccRxVcRxCallback** rxCallback, unsigned int  
minimumReplenishCount, **IxAtmConnId** \*connIdPtr, **IxAtmNpeRxVcId** \*npeVcIdPtr)  
*Connect to a Aal Pdu receive service for a particular port/vpi/vci, and service type.*

**PUBLIC**  
**IX\_STATUS ixAtmdAccRxVcFreeReplenish** (IxAtmConnId connId, IX\_MBUF \*mbufPtr)  
*Provide free mbufs for data reception on a connection.*

**PUBLIC ixAtmdAccRxVcFreeLowCallbackRegister** (IxAtmConnId connId, unsigned int  
**IX\_STATUS** numberOfMbufs, **IxAtmdAccRxVcFreeLowCallback** callback)  
*Configure the RX Free threshold value and register a callback to handle threshold notifications.*

**PUBLIC ixAtmdAccRxVcFreeEntriesQuery** (IxAtmConnId connId, unsigned int  
**IX\_STATUS** \*numberOfMbufsPtr)  
*Get the number of rx mbufs the system can accept to replenish the the rx reception mechanism on a particular channel.*

**PUBLIC**  
**IX\_STATUS ixAtmdAccRxVcEnable** (IxAtmConnId connId)  
*Start the RX service on a VC.*

**PUBLIC**  
**IX\_STATUS ixAtmdAccRxVcDisable** (IxAtmConnId connId)  
*Stop the RX service on a VC.*

**PUBLIC**  
**IX\_STATUS ixAtmdAccRxVcTryDisconnect** (IxAtmConnId connId)  
*Disconnect a VC from the RX service.*

**PUBLIC ixAtmdAccTxVcConnect** (IxAtmLogicalPort port, unsigned int vpi, unsigned int vci,  
**IX\_STATUS IxAtmdAccAalType** aalServiceType, **IxAtmdAccUserId** userId,  
**IxAtmdAccTxVcBufferReturnCallback** bufferFreeCallback, **IxAtmConnId** \*connIdPtr)

*Connect to a Aal Pdu transmit service for a particular port/vpi/vci and service type.*

**PUBLIC**  
**IX\_STATUS IxAtmdAccTxVcPduSubmit (IxAtmConnId connId, IX\_MBUF \*mbufPtr,**  
**IX\_STATUS IxAtmdAccClpStatus clp, unsigned int numberOfCells)**  
*Submit a Pdu for transmission on connection.*

**PUBLIC**  
**IX\_STATUS IxAtmdAccTxVcTryDisconnect (IxAtmConnId connId)**  
*Disconnect from a Aal Pdu transmit service for a particular port/vpi/vci.*

---

## Detailed Description

The public API for the IXP425 Atm Driver Data component.

IxAtmdAcc is the low level interface by which AAL0/AAL5 and OAM data gets transmitted to, and received from the Utopia bus.

For AAL0/AAL5 services transmit and receive connections may be established independently for unique combinations of port, VPI, and VCI.

Two AAL0 services supporting 48 or 52 byte cell data are provided. Submitted AAL0 PDUs must be a multiple of the cell data size (48/52). AAL0\_52 is a raw cell service the client must format the PDU with an ATM cell header (excluding HEC) at the start of each cell, note that AtmdAcc does not validate the cell headers in a submitted PDU.

OAM cells cannot be received over the AAL0 service but instead are received over a dedicated OAM service.

For the OAM service an "OAM Tx channel" may be enabled for a port by establishing a single dedicated OAM Tx connection on that port. A single "OAM Rx channel" for all ports may be enabled by establishing a dedicated OAM Rx connection.

The OAM service allows buffers containing 52 byte OAM F4/F5 cells to be transmitted and received over the dedicated OAM channels. HEC is appended/removed, and CRC-10 performed by the NPE. The OAM service offered by AtmdAcc is a raw cell transport service. It is assumed that ITU I.610 procedures that make use of this service are implemented above AtmdAcc.

Note that the dedicated OAM connections are established on reserved VPI, VCI, and (in the case of Rx) port values defined below. These values are used purely to discriminate the dedicated OAM channels and do not identify a particular OAM F4/F5 flow. F4/F5 flows may be realised for particular VPI/VCIs by manipulating the VPI, VCI fields of the ATM cell headers of cells in the buffers passed to AtmdAcc. Note that AtmdAcc does not validate the cell headers in a submitted OAM PDU.

This part is related to the User datapath processing

---

# Define Documentation

```
#define IX_ATMDACC_BUSY
```

Busy return code.

This constant is used to tell IxAtmDACC user that the request is correct, but cannot be processed because the IxAtmAcc resources are already used. The user has to retry its request later

Definition at line **148** of file **IxAtmdAcc.h**.

```
#define IX_ATMDACC_DEFAULT_REPLENISH_COUNT
```

Default resources usage for RxVcFree replenish mechanism.

This constant is used to tell IxAtmDACC to allocate and use the minimum of resources for rx free replenish.

*See also:*

**ixAtmdAccRxVcConnect**

Definition at line **179** of file **IxAtmdAcc.h**.

```
#define IX_ATMDACC_OAM_RX_PORT
```

The reserved dummy PORT used for all dedicated OAM Rx connections. Note that this is not a real port but must have a value that lies within the valid range of port values.

Definition at line **220** of file **IxAtmdAcc.h**.

```
#define IX_ATMDACC_OAM_RX_VCI
```

The reserved value value used for the dedicated OAM Rx connection. This value should be chosen so that it does not coincide with the VCI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VCI will fail.

Definition at line **244** of file **IxAtmdAcc.h**.

```
#define IX_ATMDACC_OAM_RX_VPI
```

The reserved value value used for the dedicated OAM Rx connection. This value should be chosen so that it does not coincide with the VPI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VPI will fail.

Definition at line **232** of file **IxAtmdAcc.h**.

```
#define IX_ATMDACC_OAM_TX_VCI
```

The reserved value used for the dedicated OAM Tx connection. This "well known" value is used by atmDacc and its clients to discriminate the OAM channel, and should be chosen so that it does not coincide with the VCI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VCI will fail.

Definition at line **208** of file **IxAtmdAcc.h**.

```
#define IX_ATMDACC_OAM_TX_VPI
```

The reserved value used for the dedicated OAM Tx connection. This "well known" value is used by atmDacc and its clients to discriminate the OAM channel, and should be chosen so that it does not coincide with the VPI value used in an AAL0/AAL5 connection. Any attempt to connect a service type other than OAM on this VPI will fail.

Definition at line **195** of file **IxAtmdAcc.h**.

```
#define IX_ATMDACC_RESOURCES_STILL_ALLOCATED
```

Disconnect return code.

This constant is used to tell IxAtmdAcc user that the disconnect functions are not complete because the resources used by the driver are not yet released. The user has to retry the disconnect call later.

Definition at line **164** of file **IxAtmdAcc.h**.

```
#define IX_ATMDACC_WARNING
```

Warning return code.

This constant is used to tell IxAtmdAcc user about a special case.

Definition at line **133** of file **IxAtmdAcc.h**.

---

## Typedef Documentation

```
typedef void(* IxAtmdAccRxVcFreeLowCallback)(IxAtmdAccUserId userId)
```

Callback prototype for free buffer level is low.

IxAtmdAccRxVcFreeLowCallback is the prototype of the user function which get called on a per-VC basis, when more mbufs are needed to continue the ATM data reception. This function is likely to supply more available mbufs by one or many calls to the replenish function ***ixAtmdAccRxVcFreeReplenish()***

This function is called when the number of available buffers for reception is going under the threshold level as defined in *ixAtmdAccRxVcFreeLowCallbackRegister()*

This function is called inside an Qmgr dispatch context. No system resource or interrupt–unsafe feature should be used inside this callback.

*See also:*

**ixAtmdAccRxVcFreeLowCallbackRegister**

**IxAtmdAccRxVcFreeLowCallback**

**ixAtmdAccRxVcFreeReplenish**

**ixAtmdAccRxVcFreeEntriesQuery**

**ixAtmdAccRxVcConnect**

*Parameters:*

*userId* (in) user Id provided in the call to *ixAtmdAccRxVcConnect()*

*Returns:*

None

Definition at line **413** of file **IxAtmdAcc.h**.

```
typedef void(* IxAtmdAccRxVcRxCallback)(IxAtmLogicalPort port, IxAtmdAccUserId userId,
IxAtmdAccPduStatus status, IxAtmdAccClpStatus clp, IX_MBUF * mbufPtr)
```

Rx callback prototype.

IxAtmdAccRxVcRxCallback is the prototype of the Rx callback user function called once per PDU to pass a receive Pdu to a user on a particular connection. The callback is likely to push the mbufs to a protocol layer, and recycle the mbufs for a further use.

*Note:*

–This function is called ONLY in the context of the *ixAtmdAccRxDispatch()* function

*See also:*

**ixAtmdAccRxDispatch**

**ixAtmdAccRxVcConnect**

*Parameters:*

*port* (in) the port on which this PDU was received a logical PHY port [IX\_UTOPIA\_PORT\_0 .. IX\_UTOPIA\_MAX\_PORTS – 1]

*userId* (in) user Id provided in the call to *ixAtmdAccRxVcConnect()*

*status* (in) an indication about the PDU validity. In the case of AAL0 the only possible value is AAL0\_VALID, in this case the client may optionally determine that an rx timeout occurred by checking if the mbuf is completely or only partially filled, the later case indicating a



timeout. In the case of OAM the only possible value is OAM valid. The status is set to *IX\_ATMDACC\_MBUF\_RETURN* when the mbuf is released during a disconnect process.

*clp* (in) clp indication for this PDU. For AAL5/AAL0\_48 this information is set if the clp bit of any rx cell is set For AAL0–52/OAM the client may inspect the CLP in individual cell headers in the PDU, and this parameter is set to 0.

*mbufPtr* (in) depending on the serve type a pointer to an mbuf (AAL5/AAL0/OAM) or mbuf chain (AAL5 only), that comprises the complete PDU data.

This parameter is guaranteed not to be a null pointer.

Definition at line **375** of file **IxAtmdAcc.h**.

```
typedef void(* IxAtmdAccTxVcBufferReturnCallback)(IxAtmdAccUserId userId, IX_MBUF * mbufPtr)
```

Buffer callback prototype.

This function is called to relinquish ownership of a transmitted buffer chain to the user.

**Note:**

–In the case of a chained mbuf the AmtdAcc component can chain many user buffers together and pass ownership to the user in one function call.

**Parameters:**

*userId* (in) user Id provided at registration of this callback.

*mbufPtr* (in) a pointer to mbufs or chain of mbufs and is guaranteed not to be a null pointer.

Definition at line **438** of file **IxAtmdAcc.h**.

```
IxAtmdAccUserId
```

User-supplied Id.

IxAtmdAccUserId is passed through callbacks and allows the IxAtmdAcc user to identify the source of a call back. The range of this user-owned Id is [0...2<sup>32</sup>–1].

The user provides this own Ids on a per-channel basis as a parameter in a call to *ixAtmdAccRxVcConnect()* or *ixAtmdAccTxVcConnect()*

**See also:**

**ixAtmdAccRxVcConnect**

**ixAtmdAccTxVcConnect**

Definition at line **327** of file **IxAtmdAcc.h**.

# Enumeration Type Documentation

## enum IxAtmdAccAalType

IxAtmdAcc AAL Service Type :.

IxAtmdAccAalType defines the type of traffic to run on this VC

### *Enumeration values:*

<i>IX_ATMDACC_AAL5</i>	ITU-T AAL5.
<i>IX_ATMDACC_AAL2</i>	ITU-T AAL2 <b>reserved</b> for future use.
<i>IX_ATMDACC_AAL0_48</i>	AAL0 48 byte payloads (cell header is added by NPE).
<i>IX_ATMDACC_AAL0_52</i>	AAL0 52 byte cell data (HEC is added by NPE).
<i>IX_ATMDACC_OAM</i>	OAM cell transport service (HEC is added by NPE).
<i>IX_ATMDACC_MAX_SERVICE_TYPE</i>	not a service, used for parameter validation

Definition at line **282** of file **IxAtmdAcc.h**.

## enum IxAtmdAccClpStatus

IxAtmdAcc CLP indication.

IxAtmdAccClpStatus defines the CLP status of the current PDU

### *Enumeration values:*

<i>IX_ATMDACC_CLP_NOT_SET</i>	CLP indication is not set.
<i>IX_ATMDACC_CLP_SET</i>	CLP indication is set.

Definition at line **303** of file **IxAtmdAcc.h**.

## enum IxAtmdAccPduStatus

IxAtmdAcc Pdu status :.

IxAtmdAccPduStatus is used during a RX operation to indicate the status of the received PDU

### *Enumeration values:*

<i>IX_ATMDACC_AAL0_VALID</i>	aal0 pdu
<i>IX_ATMDACC_OAM_VALID</i>	OAM pdu.

<i>IX_ATMDACC_AAL2_VALID</i>	aal2 pdu <b>reserved</b> for future use
<i>IX_ATMDACC_AAL5_VALID</i>	aal5 pdu complete and trailer is valid
<i>IX_ATMDACC_AAL5_PARTIAL</i>	aal5 pdu not complete, trailer is missing
<i>IX_ATMDACC_AAL5_CRC_ERROR</i>	aal5 pdu not complete, crc error/length error
<i>IX_ATMDACC_MBUF_RETURN</i>	empty buffer returned to the user

Definition at line **259** of file **IxAtmdAcc.h**.

## Function Documentation

```
ixAtmdAccInit ( void )
```

Initialise the IxAtmdAcc Component.

This function initialise the IxAtmdAcc component. This function shall be called before any other function of the API. Its role is to initialise all internal resources of the IxAtmdAcc component.

The ixQmgr component needs to be initialized prior the use of ***ixAtmdAccInit()***

### Parameters:

*none* Failing to initialize the IxAtmdAcc API before any use of it will result in a failed status. If the specified component is not present, a success status will still be returned, however, a warning indicating the NPE to download to is not present will be issued.

### Returns:

- ◇ **IX\_SUCCESS** initialisation is complete (in case of component not being present, a warning is clearly indicated)
- ◇ **IX\_FAIL** unable to process this request either because this IxAtmdAcc is already initialised or some unspecified error has occurred.

```
ixAtmdAccRxVcConnect ( IxAtmLogicalPort      port,
                      unsigned int             vpi,
                      unsigned int             vci,
                      IxAtmdAccAalType        aalServiceType,
                      IxAtmRxQueueId          rxQueueId,
                      IxAtmdAccUserId         userCallbackId,
                      IxAtmdAccRxVcRxCallback rxCallback,
                      unsigned int             minimumReplenishCount,
                      IxAtmConnId *          connIdPtr,
                      IxAtmNpeRxVcId *       npeVcIdPtr
                      )
```

Connect to a Aal Pdu receive service for a particular port/vpi/vci, and service type.

This function allows a user to connect to an Aal5/Aal0/OAM Pdu receive service for a particular port/vpi/vci. It registers the callback and allocates internal resources and a Connection Id to be used in further API calls related to this VCC.

The function will setup VC receive service on the specified rx queue.

This function is blocking and makes use internal locks, and hence should not be called from an interrupt context.

On return from ***ixAtmdAccRxVcConnect()*** with a failure status, the connection Id parameter is unspecified. Its value cannot be used. A connId is the reference by which IxAtmdAcc refers to a connected VC. This identifier is the result of a succesful call to a connect function. This identifier is invalid after a sucessful call to a disconnect function.

Calling this function for the same combination of Vpi, Vci and more than once without calling ***ixAtmdAccRxVcTryDisconnect()*** will result in a failure status.

If this function returns success the user should supply receive buffers by calling ***ixAtmdAccRxVcFreeReplenish()*** and then call ***ixAtmdAccRxVcEnable()*** to begin receiving pdus.

There is a choice of two receive Qs on which the VC pdus could be receive. The user must associate the VC with one of these. Essentially having two qs allows more flexible system configuration such as have high prioriy traffic on one q (e.g. voice) and low priority traffic on the other (e.g. data). The high priority Q could be serviced in preference to the low priority Q. One queue may be configured to be serviced as soon as there is traffic, the other queue may be configured to be serviced by a polling mechanism running at idle time.

Two AAL0 services supporting 48 or 52 byte cell data are provided. Received AAL0 PDUs will be be a multiple of the cell data size (48/52). AAL0\_52 is a raw cell service and includes an ATM cell header (excluding HEC) at the start of each cell.

A single "OAM Rx channel" for all ports may be enabled by establishing a dedicated OAM Rx connection.

The OAM service allows buffers containing 52 byte OAM F4/F5 cells to be transmitted and received over the dedicated OAM channels. HEC is appended/removed, and CRC-10 performed by the NPE. The OAM service offered by AtmdAcc is a raw cell transport service. It is assumed that ITU I.610 procedures that make use of this service are implemented above AtmdAcc.

Note that the dedicated OAM connections are established on reserved VPI,VCI, and (in the case of Rx) port values. These values are used purely to descriminate the dedicated OAM channels and do not identify a particular OAM F4/F5 flow. F4/F5 flows may be realised for particluar VPI/VCIs by manipulating the VPI,VCI fields of the ATM cell headers of cells in the buffers passed to AtmdAcc.

Calling this function prior to enable the port will fail.

*See also:*

***ixAtmdAccRxDispatch***

***ixAtmdAccRxVcEnable***

**ixAtmdAccRxVcDisable**

**ixAtmdAccRxVcTryDisconnect**

**ixAtmdAccPortEnable**

**Parameters:**

<i>port</i>	(in) VC identification : logical PHY port [IX_UTOPIA_PORT_0 .. IX_UTOPIA_MAX_PORTS – 1]
<i>vpi</i>	(in) VC identification : ATM Vpi [0..255] or IX_ATMDACC_OAM_VPI
<i>vci</i>	(in) VC identification : ATM Vci [0..65535] or IX_ATMDACC_OAM_VCI
<i>aalServiceType</i>	(in) type of service: AAL5, AAL0_48, AAL0_52, or OAM
<i>rxQueueId</i>	(in) this identifies which of two Qs the VC should use when incoming traffic is processed
<i>userCallbackId</i>	(in) user Id used later as a parameter to the supplied rxCallback.
<i>rxCallback</i>	(in) function called when mbufs are received. This parameter cannot be a null pointer.
<i>bufferFreeCallback</i>	(in) function to be called to return ownership of buffers to IxAtmdAcc user.
<i>minimumReplenishCount</i>	(in) For AAL5/AAL0 the number of free mbufs to be used with this channel. Use a high number when the expected traffic rate on this channel is high, or when the user's mbufs are small, or when the RxVcFreeLow Notification has to be invoked less often. When this value is IX_ATMDACC_DEFAULT_REPLENISH_COUNT, the minimum of resources will be used. Depending on traffic rate, pdu size and mbuf size, rxfree queue size, polling/interrupt rate, this value may require to be replaced by a different value in the range 1–128 For OAM the rxFree queue size is fixed by atmdAcc and this parameter is ignored.
<i>connIdPtr</i>	(out) pointer to a connection Id This parameter cannot be a null pointer.
<i>npeVcIdPtr</i>	(out) pointer to an npe Vc Id This parameter cannot be a null pointer.

**Returns:**

- ◇ IX\_SUCCESS successful call to IxAtmdAccRxVcConnect
- ◇ IX\_ATMDACC\_BUSY cannot process this request : no VC is available
- ◇ IX\_FAIL parameter error, VC already in use, attempt to connect AAL service on reserved OAM VPI/VCI, attempt to connect OAM service on VPI/VCI other than the reserved OAM VPI/VCI, port is not initialised, or some other error occurs during processing.

ixAtmdAccRxVcDisable ( **IxAtmConnId** *connId* )

Stop the RX service on a VC.

This function stops the traffic reception for a particular VC connection.

Once invoked, incoming Pdus are discarded by the hardware. Any Pdus pending will be freed to the user

Hence once this function returns no more receive callbacks will be called for that VC. However, buffer free callbacks will be invoked until such time as all buffers supplied by the user have been freed back to the user

Calling this function does not invalidate the `connId`. ***ixAtmdAccRxVcEnable()*** can be invoked to enable Pdu reception again.

If the traffic is already stopped, this function returns `IX_SUCCESS`.

This function is not reentrant and should not be used inside an interrupt context.

*See also:*

***ixAtmdAccRxVcConnect***

***ixAtmdAccRxVcEnable***

***ixAtmdAccRxVcDisable***

*Parameters:*

*connId* (in) connection Id as resulted from a successful call to *ixAtmdAccRxVcConnect()*

*Returns:*

◇ `IX_SUCCESS` successful call to ***ixAtmdAccRxVcDisable()***.

◇ `IX_ATMDACC_WARNING` the channel is already disabled

◇ `IX_FAIL` invalid parameters or some unspecified internal error occurred

```
ixAtmdAccRxVcEnable ( IxAtmConnId connId )
```

Start the RX service on a VC.

This function kicks-off the traffic reception for a particular VC. Once invoked, incoming PDUs will be made available by the hardware and are eventually directed to the ***ixAtmdAccRxVcRxCallback()*** callback registered for the connection.

If the traffic is already running, this function returns `IX_SUCCESS`. This function can be invoked many times.

`IxAtmdAccRxVcFreeLowCallback` event will occur only after ***ixAtmdAccRxVcEnable()*** function is invoked.

Before using this function, the ***ixAtmdAccRxVcFreeReplenish()*** function has to be used to replenish the RX Free queue. If not, incoming traffic may be discarded. and in the case of interrupt driven reception the ***ixAtmdAccRxVcFreeLowCallback()*** callback may be invoked as a side effect during a replenish action.

This function is not reentrant and should not be used inside an interrupt context.

For an VC connection this function can be called after a call to ***ixAtmdAccRxVcDisable()*** and should not be called after ***ixAtmdAccRxVcTryDisconnect()***

*See also:*

**ixAtmdAccRxVcDisable**

**ixAtmdAccRxVcConnect**

**ixAtmdAccRxVcFreeReplenish**

**Parameters:**

*connId* (in) connection Id as resulted from a succesfull call to *IxAtmdAccRxVcConnect()*

**Returns:**

◇ IX\_SUCCESS successful call to ixAtmdAccRxVcEnable

◇ IX\_ATMDACC\_WARNING the channel is already enabled

◇ IX\_FAIL invalid parameters or some unspecified internal error ocured.

```
ixAtmdAccRxVcFreeEntriesQuery ( IxAtmConnId connId,  
                                unsigned int * numberOfMbufsPtr  
                                )
```

Get the number of rx mbufs the system can accept to replenish the the rx reception mechanism on a particular channel.

The ixAtmdAccRxVcFreeEntriesQuery function is used to retrieve the current number of available mbuf entries for reception, on a per-VC basis. This function can be used to know the number of mbufs which can be provided using *ixAtmdAccRxVcFreeReplenish()*.

This function can be used from a timer context, or can be associated with a threshold event, or can be used inside an active polling mechanism which is under user control.

This function is reentrant and does not use system resources and can be invoked from an interrupt context.

**Parameters:**

*connId* (in) connection Id as resulted from a succesfull call to *IxAtmdAccRxVcConnect()*

*numberOfMbufsPtr* (out) Pointer to the number of available entries. . This parameter cannot be a null pointer.

**Returns:**

◇ IX\_SUCCESS the current number of mbufs not yet used for incoming traffic

◇ IX\_FAIL invalid parameter

*See also:*

**ixAtmdAccRxVcFreeReplenish**

```

ixAtmdAccRxVcFreeLowCallbackRegister ( IxAtmConnId connId,
                                         unsigned int numberOfMbufs,
                                         IxAtmdAccRxVcFreeLowCallback callback
                                         )

```

Configure the RX Free threshold value and register a callback to handle threshold notifications.

The function `ixAtmdAccRxVcFreeLowCallbackRegister` sets the threshold value for a particular RX VC. When the number of buffers reaches this threshold the callback is invoked.

This function should be called once per VC before RX traffic is enabled. This function will fail if the current level of the free buffers is equal or less than the threshold value.

*See also:*

**ixAtmdAccRxVcFreeLowCallbackRegister**

**IxAtmdAccRxVcFreeLowCallback**

**ixAtmdAccRxVcFreeReplenish**

**ixAtmdAccRxVcFreeEntriesQuery**

**ixAtmdAccRxVcConnect**

**Parameters:**

*connId* (in) connection Id as resulted from a successful call to `IxAtmdAccRxVcConnect()`  
*numberOfMbufs* (in) threshold number of buffers. This number has to be a power of 2, one of the values 0,1,2,4,8,16,32.... The maximum value cannot be more than half of the rxFree queue size (which can be retrieved using `ixAtmdAccRxVcFreeEntriesQuery()` before any use of the `ixAtmdAccRxVcFreeReplenish()` function)  
*callback* (in) function telling the user that the number of free buffers has reduced to the threshold value.

**Returns:**

- ◊ IX\_SUCCESS Threshold set successfully.
- ◊ IX\_FAIL parameter error or the current number of free buffers is less than or equal to the threshold supplied or some unspecified error has occurred.

**Note:**

- the callback will be called when the threshold level will drop from exactly  $(\text{numberOfMbufs} + 1)$  to  $(\text{numberOfMbufs})$ .

```

ixAtmdAccRxVcFreeReplenish ( IxAtmConnId connId,
                             IX_MBUF * mbufPtr
                             )

```



Provide free mbufs for data reception on a connection.

This function provides mbufs for data reception by the hardware. This function needs to be called by the user on a regular basis to ensure no packet loss. Providing free buffers is a connection-based feature; each connection can have different requirements in terms of buffer size number of buffers, recycling rate. This function could be invoked from within the context of a ***IxAtmdAccRxVcFreeLowCallback()*** callback for a particular VC

Mbufs provided through this function call can be chained. They will be unchained internally. A call to this function with chained mbufs or multiple calls with unchained mbufs are equivalent, but calls with unchained mbufs are more efficient.

Mbufs provided to this interface need to be able to hold at least one full cell payload (48/52 bytes, depending on service type). Chained buffers with a size less than the size supported by the hardware will be returned through the rx callback provided during the connect step.

Failing to invoke this function prior to enabling the RX traffic can result in packet loss.

This function is not reentrant for the same connId.

This function does not use system resources and can be invoked from an interrupt context.

**Note:**

- Over replenish is detected, and extra mbufs are returned through the rx callback provided during the connect step.
- Mbuf provided to the replenish function should have a length greater or equal to 48/52 bytes according to service type.
- The memory cache of mMbuf payload should be invalidated prior to Mbuf submission. Flushing the Mbuf headers is handled by IxAtmdAcc.
- When a chained mbuf is provided, this function process the mbufs up to the hardware limit and invokes the user-supplied callback to release extra buffers.

**See also:**

**ixAtmdAccRxVcFreeLowCallbackRegister**

**IxAtmdAccRxVcFreeLowCallback**

**ixAtmdAccRxVcConnect**

**Parameters:**

- |                |   |
|----------------|---|
| <i>connId</i>  | (in) connection Id as returned from a successful call to <b><i>IxAtmdAccRxVcConnect()</i></b>   |
| <i>mbufPtr</i> | (in) pointer to a mbuf structure to be used for data reception. The mbuf pointed to by this parameter can be chained to another mbuf. |

**Returns:**

- ◊ IX\_SUCCESS successful call to ***ixAtmdAccRxVcFreeReplenish()*** and the mbuf is now ready to use for incoming traffic.

- ◇ IX\_ATMDACC\_BUSY cannot process this request because the max number of outstanding free buffers has been reached or the internal resources have exhausted for this VC. The user is responsible for retrying this request later.
- ◇ IX\_FAIL cannot process this request because of parameter errors or some unspecified internal error has occurred.

**Note:**

- It is not always guaranteed the replenish step to be as fast as the hardware is consuming Rx Free mbufs. There is nothing in IxAtmdAcc to guarantee that replenish reaches the rxFree threshold level. If the threshold level is not reached, the next rxFree low notification for this channel will not be triggered. The preferred ways to replenish can be as follows (depending on applications and implementations) :
  - ◇ Replenish in a rxFree low notification until the function **ixAtmdAccRxVcFreeReplenish()** returns IX\_ATMDACC\_BUSY
  - ◇ Query the queue level using

**See also:**

- ixAtmdAccRxVcFreeEntriesQuery**, then , replenish using **ixAtmdAccRxVcFreeReplenish()**, then query the queue level again, and replenish if the threshold is still not reached.
  - ◇ Trigger replenish from an other event source and use rxFree starvation to throttle the Rx traffic.

**ixAtmdAccRxVcTryDisconnect ( IxAtmConnId connId )**

Disconnect a VC from the RX service.

This function deregisters the VC and guarantees that all resources associated with this VC are free. After its execution, the connection Id is not available.

This function will fail until such time as all resources allocated to the VC connection have been freed. The user is responsible to delay and call again this function many times until a success status is returned.

This function needs internal locks and should not be called from an interrupt context

**Parameters:**

*connId* (in) connection Id as resulted from a succesfull call to *IxAtmdAccRxVcConnect()*

**Returns:**

- ◇ IX\_SUCCESS successful call to ixAtmdAccRxVcDisable
- ◇ IX\_ATMDACC\_RESOURCES\_STILL\_ALLOCATED not all resources associated with the connection have been freed.
- ◇ IX\_FAIL cannot process this request because of a parameter error

**ixAtmdAccShow ( void )**

Show IxAtmdAcc configuration on a per port basis.

**Parameters:**

*none*

**Returns:**

*none*

**Note:**

– Display use printf() and are redirected to stdout

```
ixAtmdAccStatsReset ( void )
```

Reset all IxAtmdAcc stats.

**Parameters:**

*none*

**Returns:**

*none*

```
ixAtmdAccStatsShow ( void )
```

Show all IxAtmdAcc stats.

**Parameters:**

*none*

**Returns:**

*none*

**Note:**

– Stats display use printf() and are redirected to stdout

```
ixAtmdAccTxVcConnect ( IxAtmLogicalPort          port,
                       unsigned int                 vpi,
                       unsigned int                 vci,
                       IxAtmdAccAalType            aalServiceType,
                       IxAtmdAccUserId             userId,
                       IxAtmdAccTxVcBufferReturnCallback bufferFreeCallback,
                       IxAtmConnId *               connIdPtr
                       )
```

Connect to a Aal Pdu transmit service for a particular port/vpi/vci and service type.

This function allows a user to connect to an Aal5/Aal0/OAM Pdu transmit service for a particular port/vpi/vci. It registers the callback and allocates internal resources and a Connection Id to be used in further API calls related to this VC.

The function will setup VC transmit service on the specified on the specified port. A connId is the reference by which IxAtmdAcc refers to a connected VC. This identifier is the result of a succesful call to a connect function. This identifier is invalid after a sucessful call to a disconnect function.

This function needs internal locks, and hence should not be called from an interrupt context.

On return from **ixAtmdAccTxVcConnect()** with a failure status, the connection Id parameter is unspecified. Its value cannot be used.

Calling this function for the same combination of port, Vpi, Vci and more than once without calling **ixAtmdAccTxVcTryDisconnect()** will result in a failure status.

Two AAL0 services supporting 48 or 52 byte cell data are provided. Submitted AAL0 PDUs must be a multiple of the cell data size (48/52). AAL0\_52 is a raw cell service the client must format the PDU with an ATM cell header (excluding HEC) at the start of each cell, note that AtmdAcc does not validate the cell headers in a submitted PDU.

For the OAM service an "OAM Tx channel" may be enabled for a port by establishing a single dedicated OAM Tx connection on that port.

The OAM service allows buffers containing 52 byte OAM F4/F5 cells to be transmitted and received over the dedicated OAM channels. HEC is appended/removed, and CRC-10 performed by the NPE. The OAM service offered by AtmdAcc is a raw cell transport service. It is assumed that ITU I.610 procedures that make use of this service are implemented above AtmdAcc.

Note that the dedicated OAM connections are established on reserved VPI,VCI, and (in the case of Rx) port values. These values are used purely to descriminate the dedicated OAM channels and do not identify a particular OAM F4/F5 flow. F4/F5 flows may be realised for particluar VPI/VCIs by manipulating the VPI,VCI fields of the ATM cell headers of cells in the buffers passed to AtmdAcc.

Calling this function before enabling the port will fail.

**See also:**

**ixAtmdAccTxVcTryDisconnect**

**ixAtmdAccPortTxScheduledModeEnable**

**ixAtmdAccPortEnable**

**Parameters:**

<i>port</i>	(in) VC identification : logical PHY port [ <i>IX_UTOPIA_PORT_0</i> .. <i>IX_UTOPIA_MAX_PORTS</i> – 1]
<i>vpi</i>	(in) VC identification : ATM Vpi [0..255] or <i>IX_ATMDACC_OAM_VPI</i>
<i>vci</i>	(in) VC identification : ATM Vci [0..65535] or <i>IX_ATMDACC_OAM_VCI</i>

<i>aalServiceType</i>	(in) type of service AAL5, AAL0_48, AAL0_52, or OAM
<i>userId</i>	(in) user id to be used later during callbacks related to this channel
<i>bufferFreeCallback</i>	(in) function called when mbufs transmission is complete. This parameter cannot be a null pointer.
<i>connIdPtr</i>	(out) Pointer to a connection Id. This parameter cannot be a null pointer.

**Returns:**

- ◇ IX\_SUCCESS successful call to *IxAtmdAccRxVcConnect()*.
- ◇ IX\_ATMDACC\_BUSY cannot process this request because no VC is available
- ◇ IX\_FAIL parameter error, VC already in use, attempt to connect AAL service on reserved OAM VPI/VCI, attempt to connect OAM service on VPI/VCI other than the reserved OAM VPI/VCI, port is not initialised, or some other error occurs during processing.

**Note:**

- Unscheduled mode is not supported in ixp425 1.0. Therefore, the function *ixAtmdAccPortTxScheduledModeEnable()* need to be called for this port before any establishing a Tx Connection

```

ixAtmdAccTxVcPduSubmit ( IxAtmConnId      connId,
                        IX_MBUF *          mbufPtr,
                        IxAtmdAccClpStatus clp,
                        unsigned int       numberOfCells
                        )

```

Submit a Pdu for transmission on connection.

A data user calls this function to submit an mbufs containing a Pdu to be transmitted. The buffer supplied can be chained and the Pdu it contains must be complete.

The transmission behavior of this call depends on the operational mode of the port on which the connection is made.

In unscheduled mode the mbuf will be submitted to the hardware immediately if sufficient resource is available. Otherwise the function will return failure.

In scheduled mode the buffer is queued internally in IxAtmdAcc. The cell demand is made known to the traffic shaping entity. Cells from the buffers are MUXed onto the port some time later as dictated by the traffic shaping entity. The traffic shaping entity does this by sending transmit schedules to IxAtmdAcc via *ixAtmdAccPortTxProcess()* function call.

Note that the dedicated OAM channel is scheduled just like any other channel. This means that any OAM traffic relating to an active AAL0/AAL5 connection will be scheduled independantly of the AAL0/AAL5 traffic for that connection.

When transmission is complete, the TX Done mechanism will give the owmnershhip of these buffers back to the customer. The tx done mechanism must be in operation before transmission is attempted.

For AAL0/OAM submitted AAL0 PDUs must be a multiple of the cell data size (48/52). AAL0\_52 and OAM are raw cell services, and the client must format the PDU with an ATM cell header (excluding HEC) at the start of each cell, note that AtmdAcc does not validate the cell headers in a submitted PDU.

*See also:*

**IxAtmdAccTxVcBufferReturnCallback**

**ixAtmdAccTxDoneDispatch**

**Parameters:**

*connId* (in) connection Id as resulted from a succesfull call to **ixAtmdAccTxVcConnect()**  
*mbufPtr* (in) pointer to a chained structure of mbufs to transmit. This parameter cannot be a null pointer.  
*clp* (in) clp indication for this PDU. All cells of this pdu will be sent with the clp bit set  
*numberOfCells* (in) number of cells in the PDU.

**Returns:**

- ◇ IX\_SUCCESS successful call to **ixAtmdAccTxVcPduSubmit()** The pdu pointed by the mbufPtr parameter will be transmitted
- ◇ IX\_ATMDACC\_BUSY unable to process this request because internal resources are all used. The caller is responsible for retrying this request later.
- ◇ IX\_FAIL unable to process this request because of error in the parameters (wrong connId supplied, or wrong mbuf pointer supplied), the total length of all buffers in the chain should be a multiple of the cell size ( 48/52 depending on the service type ), or unspecified error during processing

**Note:**

- This function is not re-entrant for the same VC (e.g. : two thread cannot send PDUs for the same VC). But two threads can safely call this function with a different connection Id
- In unscheduled mode, this function is not re-entrant on a per port basis. The size of pdus is limited to 8Kb.
- 0-length mbufs should be removed from the chain before submission. The total length of the pdu (sdu + padding +trailer) has to be updated in the header of the first mbuf of a chain of mbufs.
- Aal5 trailer information (UUI, CPI, SDU length) has to be supplied before submission.
- The payload memory cache should be flushed, if needed, prior to transmission. Mbuf headers are flushed by IxAtmdAcc
- This function does not use system resources and can be used inside an interrupt context

**ixAtmdAccTxVcTryDisconnect ( IxAtmConnId connId )**

Disconnect from a Aal Pdu transmit service for a particular port/vpi/vci.

This function deregisters the VC and guarantees that all resources associated with this VC are free. After its execution, the connection Id is not available.

This function will fail until such time as all resources allocated to the VC connection have been freed. The user is responsible to delay and call again this function many times until a success status is returned.

After its execution, the connection Id is not available.

**Parameters:**

*connId* (in) connection Id as resulted from a succesfull call to *ixAtmdAccTxVcConnect()*

**Returns:**

- ◇ IX\_SUCCESS successful call to *ixAtmdAccTxVcTryDisconnect()*
- ◇ IX\_ATMDACC\_RESOURCES\_STILL\_ALLOCATED not all resources associated with the connection have been freed. This condition will disappear after Tx and TxDone is complete for this channel.
- ◇ IX\_FAIL unable to process this request because of errors in the parameters (wrong connId supplied)

**Note:**

- This function needs internal locks and should not be called from an interrupt context
- If the *IX\_ATMDACC\_RESOURCES\_STILL\_ALLOCATED* error does not clear after a while, this may be linked to a previous problem of cell overscheduling. Disabling the port and retry a disconnect will free the resources associated with this channel.

**See also:**

**ixAtmdAccPortTxProcess**

# IXP425 ATM Driver Access (IxAtmdAcc) Control API

The public API for the IXP425 Atm Driver Control component.

## Modules

**IXP425 ATM Driver Access (IxAtmdAcc) Utopia  
Control API**

*The public API for the IXP425 Atm Driver Control component.*

## Defines

**#define IX\_ATMDACC\_PORT\_DISABLE\_IN\_PROGRESS**  
*Port enable return code.*

**#define IX\_ATMDACC\_ALLPDUS**  
*All PDUs.*

## Typedefs

**typedef IxAtmdAccRxDispatcher** (IxAtmRxQueueId rxQueueId, unsigned int  
IX\_STATUS(\* numberOfPdusToProcess, unsigned int \*reservedPtr)  
*Callback prototype for notification of available PDUs for an Rx Q.*

**typedef IxAtmdAccTxDoneDispatcher** (unsigned int numberOfPdusToProcess, unsigned int  
IX\_STATUS(\* \*reservedPtr)  
*Callback prototype for transmitted mbuf when threshold level is crossed.*

**typedef void(\* IxAtmdAccPortTxLowCallback)** (IxAtmLogicalPort port, unsigned int  
numberOfAvailableCells)  
*Notification that the threshold number of scheduled cells remains in a port's transmit Q.*

**typedef IxAtmdAccTxVcDemandUpdateCallback** (IxAtmLogicalPort port, int vcId, unsigned  
IX\_STATUS(\* int numberOfCells)  
*Prototype to submit cells for transmission.*

**typedef void(\* IxAtmdAccTxVcDemandClearCallback)** (IxAtmLogicalPort port, int vcId)  
*prototype to remove all currently queued cells from a registered VC*

**typedef IxAtmdAccTxSchVcIdGetCallback** (IxAtmLogicalPort port, unsigned int vpi, unsigned  
IX\_STATUS(\* int vci, IxAtmConnId connId, int \*vcId)  
*prototype to get a scheduler vc id*



## Functions

**PUBLICIxAtmdAccRxDispatcherRegister** (**IxAtmRxQueueId** queueId, **IX\_STATUS** **IxAtmdAccRxDispatcher** callback)  
*Register a notification callback to be invoked when there is at least one entry on a particular Rx queue.*

**PUBLICIxAtmdAccRxDispatch** (**IxAtmRxQueueId** rxQueueId, unsigned int numberOfPdusToProcess, unsigned int \*numberOfPdusProcessedPtr)  
*Control function which executes Rx processing for a particular Rx stream.*

**PUBLICIxAtmdAccRxLevelQuery** (**IxAtmRxQueueId** rxQueueId, unsigned int **IX\_STATUS** \*numberOfPdusPtr)  
*Query the number of entries in a particular RX queue.*

**PUBLICIxAtmdAccRxQueueSizeQuery** (**IxAtmRxQueueId** rxQueueId, unsigned int **IX\_STATUS** \*numberOfPdusPtr)  
*Query the size of a particular RX queue.*

**PUBLICIxAtmdAccPortTxFreeEntriesQuery** (**IxAtmLogicalPort** port, unsigned int **IX\_STATUS** \*numberOfCellsPtr)  
*Get the number of available cells the system can accept for transmission.*

**PUBLICIxAtmdAccPortTxCallbackRegister** (**IxAtmLogicalPort** port, unsigned int **IX\_STATUS** numberOfCells, **IxAtmdAccPortTxLowCallback** callback)  
*Configure the Tx port threshold value and register a callback to handle threshold notifications.*

**PUBLICIxAtmdAccPortTxScheduledModeEnable** (**IxAtmLogicalPort** port, **IX\_STATUS** **IxAtmdAccTxVcDemandUpdateCallback** vcDemandUpdateCallback, **IxAtmdAccTxVcDemandClearCallback** vcDemandClearCallback, **IxAtmdAccTxSchVcIdGetCallback** vcIdGetCallback)  
*Put the port into Scheduled Mode.*

**PUBLICIxAtmdAccPortTxProcess** (**IxAtmLogicalPort** port, **IxAtmScheduleTable** **IX\_STATUS** \*scheduleTablePtr)  
*Transmit queue cells to the H/W based on the supplied schedule table.*

**PUBLICIxAtmdAccTxDoneDispatch** (unsigned int numberOfPdusToProcess, unsigned int **IX\_STATUS** \*numberOfPdusProcessedPtr)  
*Process a number of pending transmit done pdus from the hardware.*

**PUBLIC**  
**IX\_STATUS** **ixAtmdAccTxDoneLevelQuery** (unsigned int \*numberOfPdusPtr)  
*Query the current number of transmit pdus ready for recycling.*

**PUBLIC**  
**IX\_STATUS** **ixAtmdAccTxDoneQueueSizeQuery** (unsigned int \*numberOfPdusPtr)

*Query the TxDone queue size.*

**PUBLIC IxAtmdAccTxDoneDispatcherRegister** (unsigned int numberOfPdus,  
IX\_STATUS **IxAtmdAccTxDoneDispatcher** notificationCallback)  
*Configure the Tx Done stream threshold value and register a callback to handle threshold notifications.*

**PUBLIC IxAtmdAccUtopiaConfigSet** (const **IxAtmdAccUtopiaConfig**  
IX\_STATUS \*ixAtmdAccUtopiaConfigPtr)  
*Send the configuration structure to the Utopia interface.*

**PUBLIC**  
IX\_STATUS **ixAtmdAccUtopiaStatusGet** (**IxAtmdAccUtopiaStatus** \*ixAtmdAccUtopiaStatus)  
*Get the Utopia interface configuration.*

**PUBLIC**  
IX\_STATUS **ixAtmdAccPortEnable** (**IxAtmLogicalPort** port)  
*enable a PHY logical port*

**PUBLIC**  
IX\_STATUS **ixAtmdAccPortDisable** (**IxAtmLogicalPort** port)  
*disable a PHY logical port*

**PUBLIC BOOL ixAtmdAccPortDisableComplete** (**IxAtmLogicalPort** port)  
*disable a PHY logical port*

---

## Detailed Description

The public API for the IXP425 Atm Driver Control component.

IxAtmdAcc is the low level interface by which AAL PDU get transmitted to, and received from the Utopia bus

This part is related to the Control configuration

---

## Define Documentation

```
#define IX_ATMDACC_ALLPDUS
```

All PDUs.

This constant is used to tell IxAtmdAcc to process all PDUs from the RX queue or the TX Done

*See also:*

**IxAtmdAccRxDispatcher**

**IxAtmdAccTxDoneDispatcher**

Definition at line **110** of file **IxAtmdAccCtrl.h**.

```
#define IX_ATMDACC_PORT_DISABLE_IN_PROGRESS
```

Port enable return code.

This constant is used to tell IxAtmdAcc user that the port disable functions are not complete. The user can call **ixAtmdAccPortDisableComplete()** to find out when the disable has finished. The port enable can then proceed.

Definition at line **93** of file **IxAtmdAccCtrl.h**.

---

## Typedef Documentation

```
typedef void(* IxAtmdAccPortTxLowCallback)(IxAtmLogicalPort port, unsigned int  
numberOfAvailableCells)
```

Notification that the threshold number of scheduled cells remains in a port's transmit Q.

This is the prototype for the user notification function which gets called on a per-port basis, when the number of remaining scheduled cells to be transmitted decreases to the threshold level. The number of cells passed as a parameter can be used for scheduling purposes as the maximum number of cells that can be passed in a schedule table to the **ixAtmdAccPortTxProcess()** function.

*See also:*

**ixAtmdAccPortTxCallbackRegister**

**ixAtmdAccPortTxProcess**

**ixAtmdAccPortTxFreeEntriesQuery**

*Parameters:*

*port* (in) logical PHY port [**IX\_UTOPIA\_PORT\_0** ..  
**IX\_UTOPIA\_MAX\_PORTS** – 1]  
*numberOfAvailableCells* (in) number of available cell entries for the port

*Note:*

– This functions shall not use system resources when used inside an interrupt context.

Definition at line **224** of file **IxAtmdAccCtrl.h**.

```
typedef IX_STATUS(* IxAtmdAccRxDispatcher)(IxAtmRxQueueId rxQueueId, unsigned int  
numberOfPdusToProcess, unsigned int *reservedPtr)
```

Callback prototype for notification of available PDUs for an Rx Q.

This a protoype for a function which is called when there is at least one Pdu available for processing on a particular Rx Q.

This function should call *ixAtmdAccRxDispatch()* with the appropriate number of parameters to read and process the Rx Q.

**See also:**

**ixAtmdAccRxDispatch**

**ixAtmdAccRxVcConnect**

**ixAtmdAccRxDispatcherRegister**

**Parameters:**

<i>rxQueueId</i>	(in) indicates which RX queue to has Pdus to process.
<i>numberOfPdusToProcess</i>	(in) indicates the minimum number of PDUs available to process all PDUs from the queue.
<i>reservedPtr</i>	(out) pointer to a int location which can be written to, but does not retain written values. This is provided to make this prototype compatible with <i>ixAtmdAccRxDispatch()</i>

**Returns:**

◇ int – ignored.

Definition at line **144** of file **IxAtmdAccCtrl.h**.

```
typedef IX_STATUS(* IxAtmdAccTxDoneDispatcher)(unsigned int numberOfPdusToProcess, unsigned int *reservedPtr)
```

Callback prototype for transmitted mbuf when threshold level is crossed.

IxAtmdAccTxDoneDispatcher is the prototype of the user function which get called when pdus are completely transmitted. This function is likely to call the *ixAtmdAccTxDoneDispatch()* function.

This function is called when the number of available pdus for reception is crossing the threshold level as defined in *ixAtmdAccTxDoneDispatcherRegister()*

This function is called inside an Qmgr dispatch context. No system resource or interrupt–unsafe feature should be used inside this callback.

Transmitted buffers recycling implementation is a sytem–wide mechanism and needs to be set before any traffic is started. If this threshold mechanism is not used, the user is responsible for polling the transmitted buffers with *ixAtmdAccTxDoneDispatch()* and *ixAtmdAccTxDoneLevelQuery()* functions.

**See also:**

**ixAtmdAccTxDoneDispatcherRegister**

## **ixAtmdAccTxDoneDispatch**

## **ixAtmdAccTxDoneLevelQuery**

### **Parameters:**

*numberOfPdusToProcess* (in) The current number of pdus currently available for recycling  
*reservedPtr* (out) pointer to a int location which can be written to but does not retain written values. This is provided to make this prototype compatible with **ixAtmdAccTxDoneDispatch()**

### **Returns:**

- ◇ IX\_SUCCESS This is provided to make this prototype compatible with **ixAtmdAccTxDoneDispatch()**
- ◇ IX\_FAIL invalid parameters or some unspecified internal error occurred. This is provided to make this prototype compatible with **ixAtmdAccTxDoneDispatch()**

Definition at line **195** of file **IxAtmdAccCtrl.h**.

```
typedef IX_STATUS(* IxAtmdAccTxSchVcIdGetCallback)(IxAtmLogicalPort port, unsigned int vpi,  
unsigned int vci, IxAtmConnId connId, int *vcId)
```

prototype to get a scheduler vc id

IxAtmdAccTxSchVcIdGetCallback is the prototype of the function to get a scheduler vcId

### **See also:**

**IxAtmdAccTxVcDemandUpdateCallback**

**IxAtmdAccTxVcDemandClearCallback**

**IxAtmdAccTxSchVcIdGetCallback**

**ixAtmdAccPortTxScheduledModeEnable**

### **Parameters:**

*port* (in) Specifies the ATM logical port on which the VC is established  
*vpi* (in) For AAL0/AAL5 specifies the ATM vpi on which the VC is established. For OAM specifies the dedicated "OAM Tx channel" VPI.  
*vci* (in) For AAL0/AAL5 specifies the ATM vci on which the VC is established. For OAM specifies the dedicated "OAM Tx channel" VCI.  
*connId* (in) specifies the IxAtmdAcc connection Id already associated with this VC  
*vcId* (out) pointer to a vcId

### **Returns:**

- ◇ IX\_SUCCESS the function is returning a Scheduler vcId for this VC
- ◇ IX\_FAIL the function cannot process scheduling for this VC. the contents of vcId is unspecified

Definition at line **324** of file **IxAtmdAccCtrl.h**.

```
typedef void(* IxAtmdAccTxVcDemandClearCallback)(IxAtmLogicalPort port, int vcId)
```

prototype to remove all currently queued cells from a registered VC

**IxAtmdAccTxVcDemandClearCallback** is the prototype of the function to remove all currently queued cells from a registered VC. The pending cell count for the specified VC is reset to zero. After the use of this callback, the scheduler shall not schedule more cells for this VC.

This callback function is called during a VC disconnection **ixAtmdAccTxVcTryDisconnect()**

*See also:*

**IxAtmdAccTxVcDemandUpdateCallback**

**IxAtmdAccTxVcDemandClearCallback**

**IxAtmdAccTxSchVcIdGetCallback**

**ixAtmdAccPortTxScheduledModeEnable**

**ixAtmdAccTxVcTryDisconnect**

*Parameters:*

*port* (in) Specifies the ATM port on which the VC to be cleared is established

*vcId* (in) Identifies the VC to be cleared. This is the value returned by the **IxAtmdAccTxSchVcIdGetCallback()** call .

*Returns:*

none

Definition at line **289** of file **IxAtmdAccCtrl.h**.

```
typedef IX_STATUS(* IxAtmdAccTxVcDemandUpdateCallback)(IxAtmLogicalPort port, int vcId, unsigned int numberOfCells)
```

Prototype to submit cells for transmission.

**IxAtmdAccTxVcDemandUpdateCallback** is the prototype of the callback function used by AtmD to notify an ATM Scheduler that the user of a VC has submitted cells for transmission.

*See also:*

**IxAtmdAccTxVcDemandUpdateCallback**

**IxAtmdAccTxVcDemandClearCallback**

**IxAtmdAccTxSchVcIdGetCallback**

## **ixAtmdAccPortTxScheduledModeEnable**

### **Parameters:**

- port* (in) Specifies the ATM port on which the VC to be updated is established
- vcId* (in) Identifies the VC to be updated. This is the value returned by the ***IxAtmdAccTxSchVcIdGetCallback()*** call .
- numberOfCells* (in) Indicates how many ATM cells should be added to the queue for this VC.

### **Returns:**

- ◇ IX\_SUCCESS the function is registering the cell demand for this VC.
- ◇ IX\_FAIL the function cannot register cell for this VC : the scheduler maybe overloaded or misconfigured

Definition at line **255** of file **IxAtmdAccCtrl.h**.

---

## **Function Documentation**

**ixAtmdAccPortDisable** ( **IxAtmLogicalPort** *port* )

disable a PHY logical port

This function disable the transmission over one port.

When a port is disabled, the cell transmission to the Utopia interface is stopped.

### **Parameters:**

- port* (in) logical PHY port [*IX\_UTOPIA\_PORT\_0* .. *IX\_UTOPIA\_MAX\_PORTS* – 1]

### **Returns:**

- ◇ IX\_SUCCESS disable is complete
- ◇ IX\_ATMDACC\_WARNING port already disabled
- ◇ IX\_FAIL disable failed, wrong parameter .

### **Note:**

- This function needs internal locks and should not be called from an interrupt context
- The response from hardware is done through the txDone mechanism to ensure the synchronisation with tx resources. Therefore, the txDone mechanism needs to be serviced to make a PortDisable complete.

### **See also:**

**ixAtmdAccPortEnable**

**ixAtmdAccPortDisableComplete**

## **ixAtmdAccTxDoneDispatch**

**ixAtmdAccPortDisableComplete** ( **IxAtmLogicalPort** *port* )

disable a PHY logical port

This function indicates if the port disable for a port has completed. This function will return TRUE if the port has never been enabled.

### **Parameters:**

*port* (in) logical PHY port [*IX\_UTOPIA\_PORT\_0* .. *IX\_UTOPIA\_MAX\_PORTS* – 1]

### **Returns:**

- ◇ TRUE disable is complete
- ◇ FALSE disable failed, wrong parameter .

### **Note:**

- This function needs internal locks and should not be called from an interrupt context

### **See also:**

**ixAtmdAccPortEnable**

**ixAtmdAccPortDisable**

**ixAtmdAccPortEnable** ( **IxAtmLogicalPort** *port* )

enable a PHY logical port

This function enables the transmission over one port. It should be called before accessing any resource from this port and before the establishment of a VC.

When a port is enabled, the cell transmission to the Utopia interface is started. If there is no traffic already running, idle cells are sent over the interface.

This function can be called multiple times.

### **Parameters:**

*port* (in) logical PHY port [*IX\_UTOPIA\_PORT\_0* .. *IX\_UTOPIA\_MAX\_PORTS* – 1]

### **Returns:**

- ◇ IX\_SUCCESS enable is complete
- ◇ IX\_ATMDACC\_WARNING port already enabled
- ◇ IX\_FAIL enable failed, wrong parameter, or cannot initialise this port (the port is maybe already in use, or there is a hardware issue)



**Note:**

- This function needs internal locks and should not be called from an interrupt context

**See also:**

**ixAtmdAccPortDisable**

```
ixAtmdAccPortTxCallbackRegister ( IxAtmLogicalPort      port,  
                                unsigned int             numberOfCells,  
                                IxAtmdAccPortTxLowCallback callback  
                                )
```

Configure the Tx port threshold value and register a callback to handle threshold notifications.

This function sets the threshold in cells

**See also:**

**ixAtmdAccPortTxCallbackRegister**

**ixAtmdAccPortTxProcess**

**ixAtmdAccPortTxFreeEntriesQuery**

**Parameters:**

*port* (in) logical PHY port [*IX\_UTOPIA\_PORT\_0* .. *IX\_UTOPIA\_MAX\_PORTS* – 1]  
*numberOfCells* (in) threshold value which triggers the callback invocation, This number has to be one of the values 0,1,2,4,8,16,32 .... The maximum value cannot be more than half of the txVc queue size (which can be retrieved using **ixAtmdAccPortTxFreeEntriesQuery()** before any Tx traffic is sent for this port)  
*callback* (in) callback function to invoke when the threshold level is reached. This parameter cannot be a null pointer.

**Returns:**

- ◇ *IX\_SUCCESS* Successful call to **ixAtmdAccPortTxCallbackRegister()**
- ◇ *IX\_FAIL* error in the parameters, Tx channel already set for this port threshold level is not correct or within the range regarding the queue size:or unspecified error during processing:

**Note:**

- This callback function get called when the threshold level drops from (numberOfCells+1) cells to (numberOfCells) cells
- This function should be called during system initialisation, outside an interrupt context

```
ixAtmdAccPortTxFreeEntriesQuery ( IxAtmLogicalPort port,  
                                unsigned int *      numberOfCellsPtr  
                                )
```

Get the number of available cells the system can accept for transmission.

The function is used to retrieve the number of cells that can be queued for transmission to the hardware.

This number is based on the worst schedule table where one cell is stored in one schedule table entry, depending on the pdu size and mbuf size and fragmentation.

This function doesn't use system resources and can be used from a timer context, or can be associated with a threshold event, or can be used inside an active polling mechanism

**Parameters:**

*port* (in) logical PHY port [IX\_UTOPIA\_PORT\_0 .. IX\_UTOPIA\_MAX\_PORTS – 1]  
*numberOfCellsPtr* (out) number of available cells. This parameter cannot be a null pointer.

**See also:**

**ixAtmdAccPortTxProcess**

**Returns:**

- ◊ IX\_SUCCESS *numberOfCellsPtr* contains the number of cells that can be scheduled for this port.
- ◊ IX\_FAIL error in the parameters, or some processing error occurred.

```
ixAtmdAccPortTxProcess ( IxAtmLogicalPort    port,  
                        IxAtmScheduleTable * scheduleTablePtr  
                        )
```

Transmit queue cells to the H/W based on the supplied schedule table.

This function **ixAtmdAccPortTxProcess()** process the schedule table provided as a parameter to the function. As a result cells are sent to the underlying hardware for transmission.

The schedule table is executed in its entirety or not at all. So the onus is on the caller not to submit a table containing more cells than can be transmitted at that point. The maximum numbers that can be transmitted is guaranteed to be the number of cells as returned by the function **ixAtmdAccPortTxFreeEntriesQuery()**.

When the scheduler is invoked on a threshold level, IxAtmdAcc gives the minimum number of cells (to ensure the callback will fire again later) and the maximum number of cells that **ixAtmdAccPortTxProcess()** will be able to process (assuming the ATM scheduler is able to produce the worst-case schedule table, i.e. one entry per cell).

When invoked outside a threshold level, the overall number of cells of the schedule table should be less than the number of cells returned by the **ixAtmdAccPortTxFreeEntriesQuery()** function.

After invoking the **ixAtmdAccPortTxProcess()** function, it is the user choice to query again the queue level with the function **ixAtmdAccPortTxFreeEntriesQuery()** and, depending on a new cell number, submit an other schedule table.

IxAtmdAcc will check that the number of cells in the schedule table is compatible with the current transmit

level. If the

Obsolete or invalid connection Id will be silently discarded.

This function is not reentrant for the same port.

This functions doesn't use system resources and can be used inside an interrupt context.

This function is used as a response to the hardware requesting more cells to transmit.

*See also:*

**ixAtmdAccPortTxScheduledModeEnable**

**ixAtmdAccPortTxFreeEntriesQuery**

**ixAtmdAccPortTxCallbackRegister**

**ixAtmdAccPortEnable**

*Parameters:*

*port* (in) logical PHY port [*IX\_UTOPIA\_PORT\_0* .. *IX\_UTOPIA\_MAX\_PORTS* – 1]  
*scheduleTablePtr* (in) pointer to a scheduler update table. The content of this table is not modified by this function. This parameter cannot be a null pointer.

*Returns:*

- ◇ **IX\_SUCCESS** the schedule table process is complete and cells are transmitted to the hardware
- ◇ **IX\_ATMDACC\_WARNING** : Traffic will be dropped: the schedule table exceed the hardware capacity If this error is ignored, further traffic and schedule will work correctly. Overscheduling does not occur when the schedule table does not contain more entries that the number of free entries returned by **ixAtmdAccPortTxFreeEntriesQuery()**. However, Disconnect attempts just after this error will fail permanently with the error code **IX\_ATMDACC\_RESOURCES\_STILL\_ALLOCATED**, and it is necessary to disable the port to make **ixAtmdAccTxVcTryDisconnect()** successful.
- ◇ **IX\_FAIL** a wrong parameter is supplied, or the format of the schedule table is invalid, or the port is not Enabled, or an internal severe error occurred. No cells is transmitted to the hardware

*Note:*

- If the failure is linked to an overschedule of data cells the result is an inconsistency in the output traffic (one or many cells may be missing and the traffic contract is not respected).

```
ixAtmdAccPortTxScheduledModeEnable ( IxAtmLogicalPort port,  
                                     IxAtmdAccTxVcDemandUpdateCallback vcDemandUpdateCallback,  
                                     IxAtmdAccTxVcDemandClearCallback vcDemandClearCallback,  
                                     IxAtmdAccTxSchVcIdGetCallback vcIdGetCallback  
                                     )
```

Put the port into Scheduled Mode.

This function puts the specified port into scheduled mode of transmission which means an external s/w entity controls the transmission of cells on this port. This facilitates traffic shaping on the port.

Any buffers submitted on a VC for this port will be queued in IxAtmdAcc. The transmission of these buffers to and by the hardware will be driven by a transmit schedule submitted regularly in calls to **ixAtmdAccPortTxProcess()** by traffic shaping entity.

The transmit schedule is expected to be dynamic in nature based on the demand in cells for each VC on the port. Hence the callback parameters provided to this function allow IxAtmdAcc to inform the shaping entity of demand changes for each VC on the port.

By default a port is in Unscheduled Mode so if this function is not called, transmission of data is done without scheduling rules, on a first-come, first-out basis.

Once a port is put in scheduled mode it cannot be reverted to un-scheduled mode. Note that unscheduled mode is not supported in ixp425 1.0

**Note:**

- This function should be called before any VCs have been connected on a port. Otherwise this function call will return failure.
- This function uses internal locks and should not be called from an interrupt context

**See also:**

**IxAtmdAccTxVcDemandUpdateCallback**

**IxAtmdAccTxVcDemandClearCallback**

**IxAtmdAccTxSchVcIdGetCallback**

**ixAtmdAccPortTxProcess**

**Parameters:**

- |                               |   |
|-------------------------------|---|
| <i>port</i>                   | (in) logical PHY port [ <i>IX_UTOPIA_PORT_0</i> .. <i>IX_UTOPIA_MAX_PORTS</i> – 1]  |
| <i>vcDemandUpdateCallback</i> | (in) callback function used to update the number of outstanding cells for transmission. This parameter cannot be a null pointer.                                  |
| <i>vcDemandClearCallback</i>  | (in) callback function used to remove all clear the number of outstanding cells for a VC. This parameter cannot be a null pointer.                                |
| <i>vcIdGetCallback</i>        | (in) callback function used to exchange vc Identifiers between IxAtmdAcc and the entity supplying the transmit schedule. This parameter cannot be a null pointer. |

**Returns:**

- ◇ *IX\_SUCCESS* scheduler registration is complete and the port is now in scheduled mode.
- ◇ *IX\_FAIL* failed (wrong parameters, or traffic is already enabled on this port, possibly without ATM shaping)

```

ixAtmdAccRxDispatch ( IxAtmRxQueueId rxQueueId,
                      unsigned int      numberOfPdusToProcess,
                      unsigned int *    numberOfPdusProcessedPtr
                      )

```

Control function which executes Rx processing for a particular Rx stream.

The *IxAtmdAccRxDispatch()* function is used to process received Pdus available from one of the two incoming RX streams. When this function is invoked, the incoming traffic (up to the number of PDUs passed as a parameter) will be transferred to the IxAtmdAcc users through the callback ***IxAtmdAccRxVcRxCallback()***, as registered during the ***ixAtmdAccRxVcConnect()*** call.

The user receive callbacks will be executed in the context of this function.

Failing to use this function on a regular basis when there is traffic will block incoming traffic and can result in Pdus being dropped by the hardware.

This should be used to control when received pdus are handed off from the hardware to Aal users from a particular stream. The function can be used from a timer context, or can be registered as a callback in response to an rx stream threshold event, or can be used inside an active polling mechanism which is under user control.

**Note:**

- The signature of this function is directly compatible with the callback prototype which can be register with ***ixAtmdAccRxDispatcherRegister()***.

**See also:**

**ixAtmdAccRxDispatcherRegister**

**IxAtmdAccRxVcRxCallback**

**ixAtmdAccRxVcFreeEntriesQuery**

**Parameters:**

<i>rxQueueId</i>	(in) indicates which RX queue to process.
<i>numberOfPdusToProcess</i>	(in) indicates the maximum number of PDU to remove from the RX queue. A value of IX_ATMDACC_ALLPDUS indicates to process all PDUs from the queue. This includes at least the PDUs in the queue when the function is invoked. Because of real-time constraints, there is no guarantee that the queue will be empty when the function exits. If this parameter is greater than the number of entries of the queues, the function will succeed and the parameter <i>numberOfPdusProcessedPtr</i> will reflect the exact number of PDUs processed.
<i>numberOfPdusProcessedPtr</i>	(out) indicates the actual number of PDU processed during this call. This parameter cannot be a null pointer.

**Returns:**

- ◊ IX\_SUCCESS the number of PDUs as indicated in *numberOfPdusProcessedPtr* are removed from the RX queue and the VC callback are called.

◇ IX\_FAIL invalid parameters or some unspecified internal error occurred.

```
ixAtmdAccRxDispatcherRegister ( IxAtmRxQueueId      queueId,  
                                IxAtmdAccRxDispatcher callback  
                                )
```

Register a notification callback to be invoked when there is at least one entry on a particular Rx queue.

This function registers a callback to be invoked when there is at least one entry in a particular queue. The registered callback is called every time when the hardware adds one or more pdus to the specified Rx queue.

This function cannot be used when a Rx Vc using this queue is already existing.

**Note:**

–The callback function can be the API function ***ixAtmdAccRxDispatch()*** : every time the threshold level of the queue is reached, the ***ixAtmdAccRxDispatch()*** is invoked to remove all entries from the queue.

**See also:**

***ixAtmdAccRxDispatch***

***IxAtmdAccRxDispatcher***

**Parameters:**

*queueId* (in) RX queue identification

*callback* (in) function triggering the delivery of incoming traffic. This parameter cannot be a null pointer.

**Returns:**

◇ IX\_SUCCESS Successful call to ***ixAtmdAccRxDispatcherRegister()***

◇ IX\_FAIL error in the parameters, or there is an already active RX VC for this queue or some unspecified internal error occurred.

```
ixAtmdAccRxLevelQuery ( IxAtmRxQueueId rxQueueId,  
                        unsigned int *    numberOfPdusPtr  
                        )
```

Query the number of entries in a particular RX queue.

This function is used to retrieve the number of pdus received by the hardware and ready for distribution to users.

**Parameters:**

*rxQueueId* (in) indicates which of two RX queues to query.

*numberOfPdusPtr* (out) Pointer to store the number of available PDUs in the RX queue. This parameter cannot be a null pointer.

**Returns:**

- ◇ IX\_SUCCESS the value in numberOfPduPtr specifies the number of incoming pdus waiting in this queue
- ◇ IX\_FAIL an error occurs during processing. The value in numberOfPduPtr is unspecified.

**Note:**

- This function is reentrant, doesn't use system resources and can be used from an interrupt context.

```
ixAtmdAccRxQueueSizeQuery ( IxAtmRxQueueId rxQueueId,
                             unsigned int *   numberOfPduPtr
                           )
```

Query the size of a particular RX queue.

This function is used to retrieve the number of pdus the system is able to queue when reception is complete.

**Parameters:**

- rxQueueId* (in) indicates which of two RX queues to query.
- numberOfPduPtr* (out) Pointer to store the number of pdus the system is able to queue in the RX queue. This parameter cannot be a null pointer.

**Returns:**

- ◇ IX\_SUCCESS the value in numberOfPduPtr specifies the number of pdus the system is able to queue.
- ◇ IX\_FAIL an error occurs during processing. The value in numberOfPduPtr is unspecified.

**Note:**

- This function is reentrant, doesn't use system resources and can be used from an interrupt context.

```
ixAtmdAccTxDoneDispatch ( unsigned int   numberOfPduToProcess,
                           unsigned int * numberOfPduProcessedPtr
                           )
```

Process a number of pending transmit done pdus from the hardware.

As a by-product of Atm transmit operation buffers which transmission is complete need to be recycled to users. This function is invoked to service the outstanding list of transmitted buffers and pass them to VC users.

Users are handed back pdus by invoking the free callback registered during the *ixAtmdAccTxVcConnect()* call.

There is a single Tx done stream servicing all active Atm Tx ports which can contain a maximum of 64 entries. If this stream fills port transmission will stop so this function must be call sufficiently frequently to ensure no disruption to the transmit operation.

This function can be used from a timer context, or can be associated with a TxDone level threshold event (see ***ixAtmdAccTxDoneDispatcherRegister()*** ), or can be used inside an active polling mechanism under user control.

For ease of use the signature of this function is compatible with the TxDone threshold event callback prototype.

This functions can be used inside an interrupt context.

**See also:**

**ixAtmdAccTxDoneDispatcherRegister**

**IxAtmdAccTxVcBufferReturnCallback**

**ixAtmdAccTxDoneLevelQuery**

**Parameters:**

*numberOfPdusToProcess* (in) maximum number of pdus to remove from the TX Done queue  
*numberOfPdusProcessedPtr* (out) number of pdus removed from the TX Done queue. This parameter cannot be a null pointer.

**Returns:**

- ◇ IX\_SUCCESS the number of pdus as indicated in *numberOfPdusToProcess* are removed from the TX Done hardware and passed to the user through the Tx Done callback registered during a call to ***ixAtmdAccTxVcConnect()***
- ◇ IX\_FAIL invalid parameters or *numberOfPdusProcessedPtr* is a null pointer or some unspecified internal error occurred.

```
ixAtmdAccTxDoneDispatcherRegister ( unsigned int          numberOfPdus,  
                                   IxAtmdAccTxDoneDispatcher notificationCallback  
                                   )
```

Configure the Tx Done stream threshold value and register a callback to handle threshold notifications.

This function sets the threshold level in term of number of pdus at which the supplied notification function should be called.

The higher the threshold value is, the less events will be necessary to process transmitted buffers.

Transmitted buffers recycling implementation is a system-wide mechanism and needs to be set prior any traffic is started. If this threshold mechanism is not used, the user is responsible for polling the transmitted buffers thanks to ***ixAtmdAccTxDoneDispatch()*** and ***ixAtmdAccTxDoneLevelQuery()*** functions.

This function should be called during system initialisation outside an interrupt context

**See also:**

**ixAtmdAccTxDoneDispatcherRegister**



## **ixAtmdAccTxDoneDispatch**

## **ixAtmdAccTxDoneLevelQuery**

### **Parameters:**

*numberOfPdus*

(in) The number of TxDone pdus which triggers the callback invocation. This number has to be a power of 2, one of the values 0,1,2,4,8,16,32 ... The maximum value cannot be more than half of the txDone queue size (which can be retrieved using

***ixAtmdAccTxDoneQueueSizeQuery()***)

*notificationCallback*

(in) The function to invoke. (This parameter can be ***ixAtmdAccTxDoneDispatch()***). This parameter must not be a null pointer.

### **Returns:**

◇ IX\_SUCCESS Successful call to ixAtmdAccTxDoneDispatcherRegister

◇ IX\_FAIL error in the parameters:

### **Note:**

- The notificationCallback will be called exactly when the threshold level will increase from (numberOfPdus) to (numberOfPdus+1)
- If there is no Tx traffic, there is no guarantee that TxDone Pdus will be released to the user (when txDone level is permanently under the threshold level. One of the preferred way to return resources to the user is to use a mix of txDone notifications, used together with a slow rate timer and an exclusion mechanism protecting from re-entrancy
- The TxDone threshold will only hand back buffers when the threshold level is crossed. Setting this threshold to a great number reduce the interrupt rate and the cpu load, but also increase the number of outstanding mbufs and has a system wide impact when these mbufs are needed by other components.

```
ixAtmdAccTxDoneLevelQuery ( unsigned int * numberOfPdusPtr )
```

Query the current number of transmit pdus ready for recycling.

This function is used to get the number of transmitted pdus which the hardware is ready to hand back to user.

This function can be used from a timer context, or can be associated with a threshold event, or can be used inside an active polling mechanism

### **See also:**

**ixAtmdAccTxDoneDispatch**

### **Parameters:**

*numberOfPdusPtr* (out) Pointer to the number of pdus transmitted at the time of this function call,

and ready for recycling This parameter cannot be a null pointer.

**Returns:**

- ◇ IX\_SUCCESS numberOfPdusPtr contains the number of pdus ready for recycling at the time of this function call
- ◇ IX\_FAIL wrong parameter (null pointer as parameter).or unspecified rocessing error occurs..The value in numberOfPdusPtr is unspecified.

```
ixAtmdAccTxDoneQueueSizeQuery ( unsigned int * numberOfPdusPtr )
```

Query the TxDone queue size.

This function is used to get the number of pdus which the hardware is able to store after transmission is complete

The returned value can be used to set a threshold and enable a callback to be notified when the number of pdus is going over the threshold.

**See also:**

**ixAtmdAccTxDoneDispatcherRegister**

**Parameters:**

*numberOfPdusPtr* (out) Pointer to the number of pdus the system is able to queue after transmission

**Returns:**

- ◇ IX\_SUCCESS numberOfPdusPtr contains the the number of pdus the system is able to queue after transmission
- ◇ IX\_FAIL wrong parameter (null pointer as parameter).or unspecified rocessing error occurs..The value in numberOfPdusPtr is unspecified.

**Note:**

- This function is reentrant, doesn't use system resources and can be used from an interrupt context.

```
ixAtmdAccUtopiaConfigSet ( const IxAtmdAccUtopiaConfig * ixAtmdAccUtopiaConfigPtr )
```

Send the configuration structure to the Utopia interface.

This function downloads the ***IxAtmdAccUtopiaConfig*** structure to the Utopia and has the following effects

- setup the Utopia interface
- initialise the NPE
- reset the Utopia cell counters and status registers to known values

This action has to be done once at initialisation. A lock is preventing the concurrent use of ***ixAtmdAccUtopiaStatusGet()*** and ***ixAtmdAccUtopiaConfigSet()***

**Parameters:**

*ixAtmdAccNPEConfigPtr* (in) pointer to a structure to download to Utopia. This parameter cannot be a null pointer.

**Returns:**

◇ IX\_SUCCESS successful download

◇ IX\_FAIL error in the parameters, or configuration is not complete or failed

**See also:**

**ixAtmdAccUtopiaStatusGet**

```
ixAtmdAccUtopiaStatusGet ( IxAtmdAccUtopiaStatus * ixAtmdAccUtopiaStatus )
```

Get the Utopia interface configuration.

This function reads the Utopia registers and the Cell counts and fills the ***IxAtmdAccUtopiaStatus*** structure

A lock is preventing the concurrent use of ***ixAtmdAccUtopiaStatusGet()*** and ***ixAtmdAccUtopiaConfigSet()***

**Parameters:**

*ixAtmdAccUtopiaStatus* (out) pointer to structure to be updated from internal hardware counters.  
This parameter cannot be a NULL pointer.

**Returns:**

◇ IX\_SUCCESS successful read

◇ IX\_FAIL error in the parameters null pointer, or configuration read is not complete or failed

**See also:**

**ixAtmdAccUtopiaConfigSet**

# IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API

## [IXP425 ATM Driver Access (IxAtmdAcc) Control API]

The public API for the IXP425 Atm Driver Control component.

### Data Structures

struct **IxAtmdAccUtopiaConfig**  
*Utopia configuration.*

struct **IxAtmdAccUtopiaConfig::UtRxConfig\_**  
*Utopia Rx config Register.*

struct **IxAtmdAccUtopiaConfig::UtRxDefineIdle\_**  
*Utopia Rx idle cells config Register.*

struct **IxAtmdAccUtopiaConfig::UtRxEnableFields\_**  
*Utopia Rx enable Register.*

struct **IxAtmdAccUtopiaConfig::UtRxStatsConfig\_**  
*Utopia Rx stats config Register.*

struct **IxAtmdAccUtopiaConfig::UtRxTransTable0\_**  
*Utopia Rx translation table Register.*

struct **IxAtmdAccUtopiaConfig::UtRxTransTable1\_**  
*Utopia Rx translation table Register.*

struct **IxAtmdAccUtopiaConfig::UtRxTransTable2\_**  
*Utopia Rx translation table Register.*

struct **IxAtmdAccUtopiaConfig::UtRxTransTable3\_**  
*Utopia Rx translation table Register.*

struct **IxAtmdAccUtopiaConfig::UtRxTransTable4\_**  
*Utopia Rx translation table Register.*

struct **IxAtmdAccUtopiaConfig::UtRxTransTable5\_**  
*Utopia Rx translation table Register.*

struct **IxAtmdAccUtopiaConfig::UtSysConfig\_**  
*NPE setup Register.*

struct **IxAtmdAccUtopiaConfig::UtTxConfig\_**  
*Utopia Tx Config Register.*

```

struct IxAtmdAccUtopiaConfig::UtTxDefineIdle_
    Utopia Tx idle cells Register.

struct IxAtmdAccUtopiaConfig::UtTxEnableFields_
    Utopia Tx ienable fields Register.

struct IxAtmdAccUtopiaConfig::UtTxStatsConfig_
    Utopia Tx stats Register.

struct IxAtmdAccUtopiaConfig::UtTxTransTable0_
    Utopia Tx translation table Register.

struct IxAtmdAccUtopiaConfig::UtTxTransTable1_
    Utopia Tx translation table Register.

struct IxAtmdAccUtopiaConfig::UtTxTransTable2_
    Utopia Tx translation table Register.

struct IxAtmdAccUtopiaConfig::UtTxTransTable3_
    Utopia Tx translation table Register.

struct IxAtmdAccUtopiaConfig::UtTxTransTable4_
    Utopia Tx translation table Register.

struct IxAtmdAccUtopiaConfig::UtTxTransTable5_
    Utopia Tx translation table Register.

struct IxAtmdAccUtopiaStatus
    Utopia status.

struct IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_
    Utopia Rx Status Register.

struct IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_
    Utopia Tx Status Register.

```

---

## Detailed Description

The public API for the IXP425 Atm Driver Control component.

IxAtmdAcc is the low level interface by which AAL PDU get transmitted to, and received from the Utopia bus

This part is related to the UTOPIA configuration.

# IXP425 ATM Manager (IxAtmm) API

IXP425 ATM Manager component Public API.

## Data Structures

struct **IxAtmmPortCfg**

*Structure contains port-specific information required to initialize IxAtmm, and specifically, the IXP425 UTOPIA Level-2 device.*

struct **IxAtmmVc**

*This structure describes the required attributes of a virtual connection.*

## Defines

#define **IX\_ATMM\_RET\_ALREADY\_INITIALIZED**

*Component has already been initialized.*

#define **IX\_ATMM\_RET\_INVALID\_PORT**

*Specified port does not exist or is out of range.*

#define **IX\_ATMM\_RET\_INVALID\_VC\_DESCRIPTOR**

*The VC description does not adhere to ATM standards.*

#define **IX\_ATMM\_RET\_VC\_CONFLICT**

*The VPI/VCI values supplied are either reserved, or they conflict with a previously registered VC on this port.*

#define **IX\_ATMM\_RET\_PORT\_CAPACITY\_IS\_FULL**

*The virtual connection cannot be established on the port because the remaining port capacity is not sufficient to support it.*

#define **IX\_ATMM\_RET\_NO\_SUCH\_VC**

*No registered VC, as described by the supplied VCI/VPI or VC identifier values, exists on this port.*

#define **IX\_ATMM\_RET\_INVALID\_VC\_ID**

*The specified VC identifier is out of range.*

#define **IX\_ATMM\_RET\_INVALID\_PARAM\_PTR**

*A pointer parameter was NULL.*

#define **IX\_ATMM\_UTOPIA\_SPHY\_ADDR**

*The phy address when in SPHY mode.*

## Typedefs

**typedef IxAtmmVcChangeCallback** (IxAtmmVcChangeEvent eventType, IxAtmLogicalPort port, void(\* const IxAtmmVc \*vcChanged)  
*Callback type used with **ixAtmmVcChangeCallbackRegister** interface Defines a callback type which will be used to notify registered users of registration/deregistration events on a particular port.*

## Enumerations

**enum IxAtmmVcDirection** {  
    **IX\_ATMM\_VC\_DIRECTION\_TX**,  
    **IX\_ATMM\_VC\_DIRECTION\_RX**,  
    **IX\_ATMM\_VC\_DIRECTION\_INVALID**  
}  
*Definition for use in the **IxAtmmVc** structure. Indicates the direction of a VC.*

**enum IxAtmmVcChangeEvent** {  
    **IX\_ATMM\_VC\_CHANGE\_EVENT\_REGISTER**,  
    **IX\_ATMM\_VC\_CHANGE\_EVENT\_DEREGISTER**,  
    **IX\_ATMM\_VC\_CHANGE\_EVENT\_INVALID**  
}  
*Definition for use with **IxAtmmVcChangeCallback** callback. Indicates that the event type represented by the callback for this VC.*

**enum IxAtmmUtopiaLoopbackMode** {  
    **IX\_ATMM\_UTOPIA\_LOOPBACK\_DISABLED**,  
    **IX\_ATMM\_UTOPIA\_LOOPBACK\_ENABLED**,  
    **IX\_ATMM\_UTOPIA\_LOOPBACK\_INVALID**  
}  
*Definitions for use with interface to indicate that UTOPIA loopback should be enabled or disabled on initialisation.*

**enum IxAtmmPhyMode** {  
    **IX\_ATMM\_MPHY\_MODE**,  
    **IX\_ATMM\_SPHY\_MODE**,  
    **IX\_ATMM\_PHY\_MODE\_INVALID**  
}  
*Definitions for use with **ixAtmmUtopiaInit** interface to indicate that UTOPIA multi-phy/single-phy mode is used.*

## Functions

**IX\_STATUS ixAtmmInit** (void)  
*Interface to initialize the IxAtmm software component. Can be called once only.*

**IX\_STATUS ixAtmmUtopiaInit** (unsigned numPorts, IxAtmmPhyMode phyMode, IxAtmmPortCfg portCfgs[], IxAtmmUtopiaLoopbackMode loopbackMode)

*Interface to initialize the UTOPIA Level–2 ATM coprocessor for the specified number of physical ports. The function must be called before the `ixAtmmPortInitialize` interface can operate successfully.*

**IX\_STATUS ixAtmmPortInitialize (IxAtmLogicalPort port, unsigned txPortRate, unsigned rxPortRate)**  
*The interface is called following `ixAtmmUtopiaInit` () and before calls to any other `IxAtmm` interface. It serves to activate the registered ATM port with `IxAtmm`.*

**IX\_STATUS ixAtmmPortModify (IxAtmLogicalPort port, unsigned txPortRate, unsigned rxPortRate)**  
*A client may call this interface to change the existing port rate (expressed in bits/second) on an established ATM port.*

**IX\_STATUS ixAtmmPortQuery (IxAtmLogicalPort port, unsigned \*txPortRate, unsigned \*rxPortRate)**  
*The client may call this interface to request details on currently registered transmit and receive rates for an ATM port.*

**IX\_STATUS ixAtmmPortEnable (IxAtmLogicalPort port)**  
*The client call this interface to enable transmit for an ATM port. At initialisation, all the ports are disabled.*

**IX\_STATUS ixAtmmPortDisable (IxAtmLogicalPort port)**  
*The client call this interface to disable transmit for an ATM port. At initialisation, all the ports are disabled.*

**IX\_STATUS ixAtmmVcRegister (IxAtmLogicalPort port, IxAtmmVc \*vcToAdd, IxAtmSchedulerVcId \*vcId)**  
*This interface is used to register an ATM Virtual Connection on the specified ATM port.*

**IX\_STATUS ixAtmmVcDeregister (IxAtmLogicalPort port, IxAtmSchedulerVcId vcId)**  
*Function called by a client to deregister a VC from the system.*

**IX\_STATUS ixAtmmVcQuery (IxAtmLogicalPort port, unsigned vpi, unsigned vci, IxAtmmVcDirection direction, IxAtmSchedulerVcId \*vcId, IxAtmmVc \*vcDesc)**  
*This interface supplies information about an active VC on a particular port when supplied with the VPI, VCI and direction of that VC.*

**IX\_STATUS ixAtmmVcIdQuery (IxAtmLogicalPort port, IxAtmSchedulerVcId vcId, IxAtmmVc \*vcDesc)**  
*This interface supplies information about an active VC on a particular port when supplied with a vcId for that VC.*

**IX\_STATUS ixAtmmVcChangeCallbackRegister (IxAtmmVcChangeCallback callback)**  
*This interface is invoked to supply a function to `IxAtmm` which will be called to notify the client if a new VC is registered with `IxAtmm` or an existing VC is removed.*

**IX\_STATUS ixAtmmVcChangeCallbackDeregister (IxAtmmVcChangeCallback callback)**  
*This interface is invoked to deregister a previously supplied callback function.*

**IX\_STATUS ixAtmmUtopiaStatusShow (void)**  
*Display utopia status counters.*



IX\_STATUS **ixAtmmUtopiaCfgShow** (void)

*Display utopia information(config registers and status registers).*

---

## Detailed Description

IXP425 ATM Manager component Public API.

---

## Define Documentation

```
#define IX_ATMM_RET_ALREADY_INITIALIZED
```

Component has already been initialized.

Definition at line **76** of file **IxAtmm.h**.

```
#define IX_ATMM_RET_INVALID_PARAM_PTR
```

A pointer parameter was NULL.

Definition at line **122** of file **IxAtmm.h**.

```
#define IX_ATMM_RET_INVALID_PORT
```

Specified port does not exist or is out of range.

Definition at line **82** of file **IxAtmm.h**.

```
#define IX_ATMM_RET_INVALID_VC_DESCRIPTOR
```

The VC description does not adhere to ATM standards.

Definition at line **88** of file **IxAtmm.h**.

```
#define IX_ATMM_RET_INVALID_VC_ID
```

The specified VC identifier is out of range.

Definition at line **116** of file **IxAtmm.h**.

```
#define IX_ATMM_RET_NO_SUCH_VC
```

No registered VC, as described by the supplied VCI/VPI or VC identifier values, exists on this port.

Definition at line **110** of file **IxAtmm.h**.

```
#define IX_ATMM_RET_PORT_CAPACITY_IS_FULL
```

The virtual connection cannot be established on the port because the remaining port capacity is not sufficient to support it.

Definition at line **103** of file **IxAtmm.h**.

```
#define IX_ATMM_RET_VC_CONFLICT
```

The VPI/VCI values supplied are either reserved, or they conflict with a previously registered VC on this port.

Definition at line **95** of file **IxAtmm.h**.

```
#define IX_ATMM_UTOPIA_SPHY_ADDR
```

The phy address when in SPHY mode.

Definition at line **128** of file **IxAtmm.h**.

---

## Typedef Documentation

```
typedef void(* IxAtmmVcChangeCallback)(IxAtmmVcChangeEvent eventType, IxAtmLogicalPort port,  
const IxAtmmVc* vcChanged)
```

Callback type used with **ixAtmmVcChangeCallbackRegister** interface Defines a callback type which will be used to notify registered users of registration/deregistration events on a particular port.

### **Parameters:**

<i>IxAtmmVcChangeEvent eventType</i>	Event indicating whether the VC supplied has been added or removed
<i>IxAtmLogicalPort port</i>	Specifies the port on which the event has occurred
<i>IxAtmmVc* vcChanged</i>	Pointer to a structure which gives details of the VC which has been added or removed on the port

Definition at line **219** of file **IxAtmm.h**.

---

# Enumeration Type Documentation

## enum IxAtmmPhyMode

Definitions for use with **ixAtmmUtopiaInit** interface to indicate that UTOPIA multi-phy/single-phy mode is used.

### *Enumeration values:*

<i>IX_ATMM_MPHY_MODE</i>	Atmm phy mode mphy.
<i>IX_ATMM_SPHY_MODE</i>	Atmm phy mode sphy.
<i>IX_ATMM_PHY_MODE_INVALID</i>	Atmm phy mode invalid.

Definition at line **180** of file **IxAtmm.h**.

## enum IxAtmmUtopiaLoopbackMode

Definitions for use with interface to indicate that UTOPIA loopback should be enabled or disabled on initialisation.

### *Enumeration values:*

<i>IX_ATMM_UTOPIA_LOOPBACK_DISABLED</i>	Atmm Utopia loopback mode disabled.
<i>IX_ATMM_UTOPIA_LOOPBACK_ENABLED</i>	Atmm Utopia loopback mode enabled.
<i>IX_ATMM_UTOPIA_LOOPBACK_INVALID</i>	Atmm Utopia loopback mode invalid.

Definition at line **156** of file **IxAtmm.h**.

## enum IxAtmmVcChangeEvent

Definition for use with **IxAtmmVcChangeCallback** callback. Indicates that the event type represented by the callback for this VC.

### *Enumeration values:*

<i>IX_ATMM_VC_CHANGE_EVENT_REGISTER</i>	Atmm Vc event register.
<i>IX_ATMM_VC_CHANGE_EVENT_DEREGISTER</i>	Atmm Vc event de-register.
<i>IX_ATMM_VC_CHANGE_EVENT_INVALID</i>	Atmm Vc event invalid.

Definition at line **146** of file **IxAtmm.h**.

## enum IxAtmmVcDirection

Definition for use in the **IxAtmmVc** structure. Indicates the direction of a VC.

### *Enumeration values:*

<i>IX_ATMM_VC_DIRECTION_TX</i>	Atmm Vc direction transmit.
--------------------------------	-----------------------------

*IX\_ATMM\_VC\_DIRECTION\_RX*            Atmm Vc direction receive.  
*IX\_ATMM\_VC\_DIRECTION\_INVALID*   Atmm Vc direction invalid.

Definition at line **136** of file **IxAtmm.h**.

---

## Function Documentation

`ixAtmmInit ( void )`

Interface to initialize the IxAtmm software component. Can be called once only.

Must be called before any other IxAtmm API is called.

**Parameters:**

*none*

**Returns:**

◇ **IX\_SUCCESS** : IxAtmm has been successfully initialized. Calls to other IxAtmm interfaces may now be performed.

◇ **IX\_FAIL** : IxAtmm has already been initialized.

`ixAtmmPortDisable ( IxAtmLogicalPort port )`

The client call this interface to disable transmit for an ATM port. At initialisation, all the ports are disabled.

**Parameters:**

*IxAtmLogicalPort port*   Value identifies the port

**Returns:**

◇ **IX\_SUCCESS** : Transmission over this port is stopped.

◇ **IX\_FAIL** : The port parameter is not valid, or the port is already disabled

**Note:**

- When a port is disabled, Rx and Tx VC Connect requests will fail
- This function call does not stop RX traffic. It is supposed that this function is invoked when a serious problem is detected (e.g. physical layer broken). Then, the RX traffic is not passing.
- This function is blocking until the hw acknowledge that the transmission is stopped.
- This function uses system resources and should not be used inside an interrupt context.

**See also:**

**ixAtmmPortEnable**

```
ixAtmmPortEnable ( IxAtmLogicalPort port )
```

The client call this interface to enable transmit for an ATM port. At initialisation, all the ports are disabled.

**Parameters:**

*IxAtmLogicalPort* port Value identifies the port

**Returns:**

◇ IX\_SUCCESS : Transmission over this port is started.

◇ IX\_FAIL : The port parameter is not valid, or the port is already enabled

**Note:**

- When a port is disabled, Rx and Tx VC Connect requests will fail
- This function uses system resources and should not be used inside an interrupt context.

**See also:**

**ixAtmmPortDisable**

```
ixAtmmPortInitialize ( IxAtmLogicalPort port,  
                      unsigned          txPortRate,  
                      unsigned          rxPortRate  
                      )
```

The interface is called following **ixAtmmUtopiaInit** () and before calls to any other IxAtmm interface. It serves to activate the registered ATM port with IxAtmm.

The transmit and receive port rates are specified in bits per second. This translates to ATM cells per second according to the following formula:  $\text{CellsPerSecond} = \text{portRate} / (53 \times 8)$  The IXP425 device supports only 53 byte cells. The client shall make sure that the off-chip physical layer device has already been initialized.

IxAtmm will configure IxAtmdAcc and IxAtmSch to enable scheduling on the port.

This interface must be called once for each active port in the system. The first time the interface is invoked, it will configure the mechanism by which the handling of transmit, transmit-done and receive are driven with the IxAtmdAcc component.

This function is reentrant.

**Note:**

The minimum tx rate that will be accepted is 424 bit/s which equates to 1 cell (53 bytes) per second.

**Parameters:**

*IxAtmLogicalPort*  
port Identifies the port which is to be initialized.

*unsigned*  
*txPortRate* Value specifies the transmit port rate for this port in bits/second. This value is used by the ATM Scheduler component is evaluating VC access requests for

the port.

*unsigned rxPortRate* Value specifies the receive port rate for this port in bits/second.

**Returns:**

- ◇ IX\_SUCCESS : The specified ATM port has been successfully initialized. IxAtmm is ready to accept VC registrations on this port.
- ◇ IX\_ATMM\_RET\_ALREADY\_INITIALIZED : ixAtmmPortInitialize has already been called successfully on this port. The current call is rejected.
- ◇ IX\_ATMM\_RET\_INVALID\_PORT : The port value indicated in the input is not valid. The request is rejected.
- ◇ IX\_FAIL : IxAtmm could not initialize the port because the inputs are not understood.

**See also:**

**ixAtmmPortEnable, ixAtmmPortDisable**

```
ixAtmmPortModify ( IxAtmLogicalPort port,
                   unsigned          txPortRate,
                   unsigned          rxPortRate
                   )
```

A client may call this interface to change the existing port rate (expressed in bits/second) on an established ATM port.

**Parameters:**

<i>IxAtmLogicalPort port</i>	Identifies the port which is to be initialized.
<i>unsigned txPortRate</i>	Value specifies the transmit port rate for this port in bits/second. This value is used by the ATM Scheduler component is evaluating VC access requests for the port.
<i>unsigned rxPortRate</i>	Value specifies the receive port rate for this port in bits/second.

**Returns:**

- ◇ IX\_SUCCESS : The indicated ATM port rates have been successfully modified.
- ◇ IX\_ATMM\_RET\_INVALID\_PORT : The port value indicated in the input is not valid. The request is rejected.
- ◇ IX\_FAIL : IxAtmm could not update the port because the inputs are not understood, or the interface was called before the port was initialized.

```

ixAtmmPortQuery ( IxAtmLogicalPort port,
                  unsigned *      txPortRate,
                  unsigned *      rxPortRate
                  )

```

The client may call this interface to request details on currently registered transmit and receive rates for an ATM port.

**Parameters:**

<i>IxAtmLogicalPort</i> port	Value identifies the port from which the rate details are requested.
<i>OUT unsigned</i> <i>*txPortRate</i>	Pointer to a value which will be filled with the value of the transmit port rate specified in bits/second.
<i>OUT unsigned</i> <i>*rxPortRate</i>	Pointer to a value which will be filled with the value of the receive port rate specified in bits/second.

**Returns:**

- ◇ IX\_SUCCESS : The information requested on the specified port has been successfully supplied in the output.
- ◇ IX\_ATMM\_RET\_INVALID\_PORT : The port value indicated in the input is not valid. The request is rejected.
- ◇ IX\_ATMM\_RET\_INVALID\_PARAM\_PTR : A pointer parameter was NULL.
- ◇ IX\_FAIL : IxAtmm could not update the port because the inputs are not understood, or the interface was called before the port was initialized.

```

ixAtmmUtopiaCfgShow ( void )

```

Display utopia information(config registers and status registers).

**Parameters:**

*none*

**Returns:**

- ◇ IX\_SUCCESS : Show function was successful
- ◇ IX\_FAIL : Internal failure

```

ixAtmmUtopiaInit ( unsigned          numPorts,
                  IxAtmmPhyMode      phyMode,
                  IxAtmmPortCfg      portCfgs[],
                  IxAtmmUtopiaLoopbackMode loopbackMode
                  )

```

Interface to initialize the UTOPIA Level-2 ATM coprocessor for the specified number of physical ports. The function must be called before the ixAtmmPortInitialize interface can operate successfully.

**Parameters:**

<i>unsigned numPorts</i>	Indicates the total number of logical ports that are active on the device. Up to 12 ports are supported.
<i>IxAtmmPhyMode phyMode</i>	Put the Utopia coprocessor in SPHY or MPHY mode.
<i>IxAtmmPortCfg portCfgs[]</i>	Pointer to an array of elements detailing the UTOPIA specific port characteristics. The length of the array must be equal to the number of ports activated. ATM ports are referred to by the relevant offset in this array in all subsequent IxAtmm interface calls.
<i>IxAtmmUtopiaLoopbackMode loopbackMode</i>	Value must be one of <b>IX_ATMM_UTOPIA_LOOPBACK_ENABLED</b> or <b>IX_ATMM_UTOPIA_LOOPBACK_DISABLED</b> indicating whether loopback should be enabled on the device. Loopback can only be supported on a single PHY, therefore the numPorts parameter must be 1 if loopback is enabled.

**Returns:**

- ◇ **IX\_SUCCESS** : Indicates that the UTOPIA device has been successfully initialized for the supplied ports.
- ◇ **IX\_ATMM\_RET\_ALREADY\_INITIALIZED** : The UTOPIA device has already been initialized.
- ◇ **IX\_FAIL** : The supplied parameters are invalid or have been rejected by the UTOPIA-NPE device.

**Warning:**

This interface may only be called once. Port identifiers are assumed to range from 0 to (numPorts – 1) in all instances. In all subsequent calls to interfaces supplied by IxAtmm, the specified port value is expected to represent the offset in the portCfgs array specified in this interface. i.e. The first port in this array will subsequently be represented as port 0, the second port as port 1, and so on.

```
ixAtmmUtopiaStatusShow ( void )
```

Display utopia status counters.

**Parameters:**

*none*

**Returns:**

- ◇ **IX\_SUCCESS** : Show function was successful
- ◇ **IX\_FAIL** : Internal failure

```
ixAtmmVcChangeCallbackDeregister ( IxAtmmVcChangeCallback callback )
```



This interface is invoked to deregister a previously supplied callback function.

**Parameters:**

<i>IxAtmmVcChangeCallback callback</i>	Callback which complies with the <b>IxAtmmVcChangeCallback</b> definition. This function will be removed from the table of callbacks.
--	---

**Returns:**

- ◇ IX\_SUCCESS : The specified callback has been deregistered successfully from IxAtmm.
- ◇ IX\_FAIL : Either the supplied callback is invalid, or is not currently registered with IxAtmm.

```
ixAtmmVcChangeCallbackRegister ( IxAtmmVcChangeCallback callback )
```

This interface is invoked to supply a function to IxAtmm which will be called to notify the client if a new VC is registered with IxAtmm or an existing VC is removed.

The callback, when invoked, will run within the context of the call to **ixAtmmVcRegister** or **ixAtmmVcDeregister** which caused the change of state.

A maximum of 32 callbacks may be registered in with IxAtmm.

**Parameters:**

<i>IxAtmmVcChangeCallback callback</i>	Callback which complies with the <b>IxAtmmVcChangeCallback</b> definition. This function will be invoked by IxAtmm with the appropriate parameters for the relevant VC when any VC has been registered or deregistered with IxAtmm.
--	---

**Returns:**

- ◇ IX\_SUCCESS : The specified callback has been registered successfully with IxAtmm and will be invoked when appropriate.
- ◇ IX\_FAIL : Either the supplied callback is invalid, or IxAtmm has already registered 32 and cannot accommodate any further registrations of this type. The request is rejected.

**Warning:**

The client must not call either the **ixAtmmVcRegister** or **ixAtmmVcDeregister** interfaces from within the supplied callback function.

```
ixAtmmVcDeregister ( IxAtmLogicalPort port,  
                    IxAtmSchedulerVcId vcId  
                    )
```

Function called by a client to deregister a VC from the system.

With the removal of each new VC from a port, a series of registered callback functions are invoked by the IxAtmm component to notify possible external components of the change. The callback functions are registered using the **ixAtmmVcChangeCallbackRegister**.

The IxAtmSch component is notified of the removal of transmit VCs.

**Parameters:**

<i>IxAtmLogicalPort</i> <i>port</i>	Identifies port on which the VC to be removed is currently registered.
<i>IxAtmSchedulerVcId</i> <i>vcId</i>	VC identifier value of the VC to be deregistered. This value was supplied to the client when the VC was originally registered. This value can also be queried from the IxAtmm component through the <b>ixAtmmVcQuery</b> interface.

**Returns:**

- ◇ IX\_SUCCESS : The specified VC has been successfully removed from this port.
- ◇ IX\_ATMM\_RET\_INVALID\_PORT : The port value indicated in the input is not valid or has not been initialized. The request is rejected.
- ◇ IX\_FAIL : There is no registered VC associated with the supplied identifier registered on this port.

```
ixAtmmVcIdQuery ( IxAtmLogicalPort    port,  
                  IxAtmSchedulerVcId vcId,  
                  IxAtmmVc *          vcDesc  
                )
```

This interface supplies information about an active VC on a particular port when supplied with a vcId for that VC.

**Parameters:**

<i>IxAtmLogicalPort</i> <i>port</i>	Identifies port on which the VC to be queried is currently registered.
<i>IxAtmSchedulerVcId</i> <i>vcId</i>	Value returned by <b>ixAtmmVcRegister</b> which uniquely identifies the requested VC on this port.
<i>OUT IxAtmmVc</i> * <i>vcDesc</i>	Pointer to an <b>IxAtmmVc</b> structure which will be filled with the specific details of the requested VC, if it exists on this port.

**Returns:**

- ◇ IX\_SUCCESS : The specified VC has been found on this port and the requested details have been returned.
- ◇ IX\_ATMM\_RET\_INVALID\_PORT : The port value indicated in the input is not valid or has not been initialized. The request is rejected.
- ◇ IX\_ATMM\_RET\_NO\_SUCH\_VC : No VC exists on the specified port which matches the supplied identifier. No data is returned.
- ◇ IX\_ATMM\_RET\_INVALID\_PARAM\_PTR : A pointer parameter was NULL.

```

ixAtmmVcQuery ( IxAtmLogicalPort    port,
                unsigned               vpi,
                unsigned               vci,
                IxAtmmVcDirection    direction,
                IxAtmSchedulerVcId * vcId,
                IxAtmmVc *           vcDesc
            )

```

This interface supplies information about an active VC on a particular port when supplied with the VPI, VCI and direction of that VC.

**Parameters:**

<i>IxAtmLogicalPort</i> port	Identifies port on which the VC to be queried is currently registered.
<i>unsigned vpi</i>	ATM VPI value of the requested VC.
<i>unsigned vci</i>	ATM VCI value of the requested VC.
<i>IxAtmmVcDirection</i> direction	One of <b>IX_ATMM_VC_DIRECTION_TX</b> or <b>IX_ATMM_VC_DIRECTION_RX</b> indicating the direction (Tx or Rx) of the requested VC.
<i>OUT IxAtmSchedulerVcId *vcId</i>	Pointer to an integer value which will be filled with the VC identifier value for the requested VC (as returned by <b>ixAtmmVcRegister</b> ), if it exists on this port.
<i>OUT IxAtmmVc *vcDesc</i>	Pointer to an <b>IxAtmmVc</b> structure which will be filled with the specific details of the requested VC, if it exists on this port.

**Returns:**

- ◇ **IX\_SUCCESS** : The specified VC has been found on this port and the requested details have been returned.
- ◇ **IX\_ATMM\_RET\_INVALID\_PORT** : The port value indicated in the input is not valid or has not been initialized. The request is rejected.
- ◇ **IX\_ATMM\_RET\_NO\_SUCH\_VC** : No VC exists on the specified port which matches the search criteria (VPI, VCI, direction) given. No data is returned.
- ◇ **IX\_ATMM\_RET\_INVALID\_PARAM\_PTR** : A pointer parameter was NULL.

```

ixAtmmVcRegister ( IxAtmLogicalPort    port,
                  IxAtmmVc *           vcToAdd,
                  IxAtmSchedulerVcId * vcId
            )

```

This interface is used to register an ATM Virtual Connection on the specified ATM port.

Each call to this interface registers a unidirectional virtual connection with the parameters specified. If a bi-directional VC is needed, the function should be called twice (once for each direction, Tx & Rx) where the VPI and VCI and port parameters in each call are identical.

With the addition of each new VC to a port, a series of callback functions are invoked by the IxAtmm component to notify possible external components of the change. The callback functions are registered

using the **IxAtmmVcChangeCallbackRegister** interface.

The IxAtmSch component is notified of the registration of transmit VCs.

**Parameters:**

<i>IxAtmLogicalPort</i> port	Identifies port on which the specified VC is to be registered.
<i>IxAtmmVc</i> *vcToAdd	Pointer to an <b>IxAtmmVc</b> structure containing a description of the VC to be registered. The client shall fill the vpi, vci and direction and relevant trafficDesc members of this structure before calling this function.
OUT	Pointer to an integer value which is filled with the per-port unique identifier value for this VC. This identifier will be required when a request is made to deregister or change this VC. VC identifiers for transmit VCs will have a value between 0–43, i.e. 32 data Tx VCs + 12 OAM Tx Port VCs. Receive VCs will have a value between 44–66, i.e. 32 data Rx VCs + 1 OAM Rx VC.
<i>IxAtmSchedulerVcId</i> *vcId	

**Returns:**

- ◇ IX\_SUCCESS : The VC has been successfully registered on this port. The VC is ready for a client to configure IxAtmdAcc for receive and transmit operations on the VC.
- ◇ IX\_ATMM\_RET\_INVALID\_PORT : The port value indicated in the input is not valid or has not been initialized. The request is rejected.
- ◇ IX\_ATMM\_RET\_INVALID\_VC\_DESCRIPTOR : The descriptor pointed to by vcToAdd is invalid. The registration request is rejected.
- ◇ IX\_ATMM\_RET\_VC\_CONFLICT : The VC requested conflicts with reserved VPI and/or VCI values or with another VC already activated on this port.
- ◇ IX\_ATMM\_RET\_PORT\_CAPACITY\_IS\_FULL : The VC cannot be registered in the port because the port capacity is insufficient to support the requested ATM traffic contract. The registration request is rejected.
- ◇ IX\_ATMM\_RET\_INVALID\_PARAM\_PTR : A pointer parameter was NULL.

**Warning:**

IxAtmm has no capability of signaling or negotiating a virtual connection. Negotiation of the admission of the VC to the network is beyond the scope of this function. This is assumed to be performed by the calling client, if appropriate, before or after this function is called.

# IXP425 ATM Transmit Scheduler (IxAtmSch) API

IXP425 ATM scheduler component Public API.

## Defines

#define **IX\_ATMSCH\_RET\_NOT\_ADMITTED**

*Indicates that CAC function has rejected VC registration due to insufficient line capacity.*

#define **IX\_ATMSCH\_RET\_QUEUE\_FULL**

*Indicates that the VC queue is full, no more demand can be queued at this time.*

#define **IX\_ATMSCH\_RET\_QUEUE\_EMPTY**

*Indicates that all VC queues on this port are empty and therefore there are no cells to be scheduled at this time.*

## Functions

**IX\_STATUS ixAtmSchInit** (void)

*This function is used to initialize the ixAtmSch component. It should be called before any other IxAtmSch API function.*

**IX\_STATUS ixAtmSchPortModelInitialize** (IxAtmLogicalPort port, unsigned int portRate, unsigned int minCellsToSchedule)

*This function shall be called first to initialize an ATM port before any other ixAtmSch API calls may be made for that port.*

**IX\_STATUS ixAtmSchPortRateModify** (IxAtmLogicalPort port, unsigned int portRate)

*This function is called to modify the portRate on a previously initialized port, typically in the event that the line condition of the port changes.*

**IX\_STATUS ixAtmSchVcModelSetup** (IxAtmLogicalPort port, IxAtmTrafficDescriptor \*trafficDesc, IxAtmSchedulerVcId \*vcId)

*A client calls this interface to set up an upstream (transmitting) virtual connection model (VC) on the specified ATM port. This function also provides the virtual \* connection admission control (CAC) service to the client.*

**IX\_STATUS ixAtmSchVcConnIdSet** (IxAtmLogicalPort port, IxAtmSchedulerVcId vcId, IxAtmConnId vcUserConnId)

*A client calls this interface to set the vcUserConnId for a VC on the specified ATM port. This vcUserConnId will default to IX\_ATM\_IDLE\_CELLS\_CONNID if this function is not called for a VC. Hence if the client does not call this function for a VC then only idle cells will be scheduled for this VC.*

**IX\_STATUS ixAtmSchVcModelRemove** (IxAtmLogicalPort port, IxAtmSchedulerVcId vcId)

*Interface called by the client to remove a previously established VC on a particular port.*

**IX\_STATUS ixAtmSchVcQueueUpdate** (**IxAtmLogicalPort** port, **IxAtmSchedulerVcId** vcId, unsigned int numberOfCells)

*The client calls this function to notify IxAtmSch that the user of a VC has submitted cells for transmission.*

**IX\_STATUS ixAtmSchVcQueueClear** (**IxAtmLogicalPort** port, **IxAtmSchedulerVcId** vcId)

*The client calls this function to remove all currently queued cells from a registered VC. The pending cell count for the specified VC is reset to zero.*

**IX\_STATUS ixAtmSchTableUpdate** (**IxAtmLogicalPort** port, unsigned int maxCells, **IxAtmScheduleTable** \*\*rettable)

*The client calls this function to request an update of the schedule table for a particular ATM port.*

**void ixAtmSchShow** (void)

*Utility function which will print statistics on the current and accumulated state of VCs and traffic in the ATM scheduler component. Output is sent to the default output device.*

**void ixAtmSchStatsClear** (void)

*Utility function which will reset all counter statistics in the ATM scheduler to zero.*

---

## Detailed Description

IXP425 ATM scheduler component Public API.

---

## Define Documentation

```
#define IX_ATMSCH_RET_NOT_ADMITTED
```

Indicates that CAC function has rejected VC registration due to insufficient line capacity.

Definition at line **88** of file **IxAtmSch.h**.

```
#define IX_ATMSCH_RET_QUEUE_EMPTY
```

Indicates that all VC queues on this port are empty and therefore there are no cells to be scheduled at this time.

Definition at line **106** of file **IxAtmSch.h**.

```
#define IX_ATMSCH_RET_QUEUE_FULL
```

Indicates that the VC queue is full, no more demand can be queued at this time.

## Function Documentation

**ixAtmSchInit** ( void )

This function is used to initialize the ixAtmSch component. It should be called before any other IxAtmSch API function.

**Parameters:**

*None*

**Returns:**

- ◇ **IX\_SUCCESS** : indicates that
  1. The ATM scheduler component has been successfully initialized.
  2. The scheduler is ready to accept Port modelling requests.
- ◇ **IX\_FAIL** : Some internal error has prevented the scheduler component from initialising.

```
ixAtmSchPortModelInitialize ( IxAtmLogicalPort port,  
                             unsigned int      portRate,  
                             unsigned int      minCellsToSchedule  
                             )
```

This function shall be called first to initialize an ATM port before any other ixAtmSch API calls may be made for that port.

**Parameters:**

- |  |   |
|--|---|
| <i>IxAtmLogicalPort</i> port           | The specific port to initialize. Valid values range from 0 to IX_UTOPIA_MAX_PORTS – 1, representing a maximum of IX_UTOPIA_MAX_PORTS possible ports.  |
| <i>unsigned int</i> portRate           | Value indicating the upstream capacity of the indicated port. The value should be supplied in units of ATM (53 bytes) cells per second. A port rate of 800Kbits/s is the equivalent of 1886 cells per second                    |
| <i>unsigned int</i> minCellsToSchedule | This parameter specifies the minimum number of cells which the scheduler will put in a schedule table for this port. This value sets the worst case CDVT for VCs on this port i.e. $CDVT = 1 * minCellsToSchedule / portRate$ . |

**Returns:**

- ◇ **IX\_SUCCESS** : indicates that
  1. The ATM scheduler has been successfully initialized.
  2. The requested port model has been established.
  3. The scheduler is ready to accept VC modelling requests on the ATM port.
- ◇ **IX\_FAIL** : indicates the requested port could not be initialized.

```
ixAtmSchPortRateModify ( IxAtmLogicalPort port,
                        unsigned int      portRate
                        )
```

This function is called to modify the portRate on a previously initialized port, typically in the event that the line condition of the port changes.

**Parameters:**

<i>IxAtmLogicalPort</i> <i>port</i>	Specifies the ATM port which is to be modified.
<i>unsigned int</i> <i>portRate</i>	Value indicating the new upstream capacity for this port in cells/second. A port rate of 800Kbits/s is the equivalent of 1886 cells per second

**Returns:**

- ◊ **IX\_SUCCESS** : The port rate has been successfully modified.
- ◊ **IX\_FAIL** : The port rate could not be modified, either because the input data was invalid, or the new port rate is insufficient to support established ATM VC contracts on this port.

**Warning:**

The IxAtmSch component will validate the supplied port rate is sufficient to support all established VC contracts on the port. If the new port rate is insufficient to support all established contracts then the request to modify the port rate will be rejected. In this event, the user is expected to remove established contracts using the ixAtmSchVcModelRemove interface and then retry this interface.

**See also:**

**ixAtmSchVcModelRemove()**

```
ixAtmSchShow ( void )
```

Utility function which will print statistics on the current and accumulated state of VCs and traffic in the ATM scheduler component. Output is sent to the default output device.

**Parameters:**

*none*

**Returns:**

*none*

```
ixAtmSchStatsClear ( void )
```

Utility function which will reset all counter statistics in the ATM scheduler to zero.

**Parameters:**

*none*

**Returns:**

*none*



```

ixAtmSchTableUpdate ( IxAtmLogicalPort      port,
                      unsigned int            maxCells,
                      IxAtmScheduleTable ** rettable
                      )

```

The client calls this function to request an update of the schedule table for a particular ATM port.

This is called when the client decides it needs a new sequence of cells to send (probably because the transmit queue is near to empty for this ATM port). The scheduler will use its stored information on the cells submitted for transmit (i.e. data supplied via **ixAtmSchVcQueueUpdate** function) with the traffic descriptor information of all established VCs on the ATM port to decide the sequence of cells to be sent and fill the schedule table for a period of time into the future.

IxAtmSch will guarantee a minimum of minCellsToSchedule if there is at least one cell ready to send. If there are no cells then IX\_ATMSCH\_RET\_QUEUE\_EMPTY is returned.

This implementation of ixAtmSchTableUpdate uses no operating system or external facilities, either directly or indirectly. This allows clients to call this function from within an FIQ interrupt handler.

**Parameters:**

<i>IxAtmLogicalPort port</i>	Specifies the ATM port for which requested schedule table is to be generated.
<i>unsigned maxCells</i>	Specifies the maximum number of cells that must be scheduled in the supplied table during any call to the interface.
<i>OUT IxAtmScheduleTable **table</i>	A pointer to an area of storage is returned which contains the generated schedule table. The client should not modify the contents of this table.

**Returns:**

- ◇ **IX\_SUCCESS** : The schedule table has been published. Currently there is at least one VC queue that is nonempty.
- ◇ **IX\_ATMSCH\_RET\_QUEUE\_EMPTY** : Currently all VC queues on this port are empty. The schedule table returned is set to NULL. The client is not expected to invoke this function again until more cells have been submitted on this port through the **ixAtmSchVcQueueUpdate** function.
- ◇ **IX\_FAIL** : The input are invalid. No action is taken.

**Warning:**

IxAtmSch assumes that the calling software ensures that calls to ixAtmSchVcQueueUpdate, ixAtmSchVcQueueClear and ixAtmSchTableUpdate are both self and mutually exclusive for the same port.

Subsequent calls to this function for the same port will overwrite the contents of previously supplied schedule tables. The client must be completely finished with the previously supplied schedule table before calling this function again for the same port.

**See also:**

**ixAtmSchVcQueueUpdate(), ixAtmSchVcQueueClear(), ixAtmSchTableUpdate().**

```

ixAtmSchVcConnIdSet ( IxAtmLogicalPort    port,
                      IxAtmSchedulerVcId  vcId,
                      IxAtmConnId         vcUserConnId
                      )

```

A client calls this interface to set the vcUserConnId for a VC on the specified ATM port. This vcUserConnId will default to IX\_ATM\_IDLE\_CELLS\_CONNID if this function is not called for a VC. Hence if the client does not call this function for a VC then only idle cells will be scheduled for this VC.

**Parameters:**

<i>IxAtmLogicalPort</i> <i>port</i>	Specifies the ATM port on which the upstream VC is has been established.
<i>IxAtmSchedulerVcId</i> <i>vcId</i>	This is the unique identifier for this virtual connection. A valid identification is a non-negative number and is all ports.
<i>IxAtmConnId</i> <i>vcUserConnId</i>	The connId is used to refer to a VC in schedule table entries. It is treated as the Id by which the scheduler client knows the VC. It is used in any communications from the Scheduler to the scheduler user e.g. schedule table entries.

**Returns:**

- ◇ **IX\_SUCCESS** : The id has successfully been set.
- ◇ **IX\_FAIL** :Input data are invalid. connId id is not established.

```

ixAtmSchVcModelRemove ( IxAtmLogicalPort    port,
                        IxAtmSchedulerVcId  vcId
                        )

```

Interface called by the client to remove a previously established VC on a particular port.

**Parameters:**

<i>IxAtmLogicalPort</i> <i>port</i>	Specifies the ATM port on which the VC to be removed is established.
<i>IxAtmSchedulerVcId</i> <i>vcId</i>	Identifies the VC to be removed. This is the value returned by the <b>ixAtmSchVcModelSetup</b> call which established the relevant VC.

**Returns:**

- ◇ **IX\_SUCCESS** : The VC has been successfully removed from this port. It is no longer modelled on this port.
- ◇ **IX\_FAIL** :Input data are invalid. The VC is still being modeled by the traffic shaper.

**See also:**

**ixAtmSchVcModelSetup()**

```

ixAtmSchVcModelSetup ( IxAtmLogicalPort    port,
                       IxAtmTrafficDescriptor * trafficDesc,
                       IxAtmSchedulerVcId *   vcId
                       )

```

A client calls this interface to set up an upstream (transmitting) virtual connection model (VC) on the specified ATM port. This function also provides the virtual \* connection admission control (CAC) service to the client.

**Parameters:**

<i>IxAtmLogicalPort</i> <i>port</i>	Specifies the ATM port on which the upstream VC is to be established.
<i>IxAtmTrafficDescriptor</i> <i>*trafficDesc</i>	Pointer to a structure describing the requested traffic contract of the VC to be established. This structure contains the typical ATM traffic descriptor values (e.g. PCR, SCR, MBS, CDVT, etc.) defined by the ATM standard.
<i>OUT IxAtmSchedulerVcId</i> <i>vcId</i>	This value will be filled with the port-unique identifier for this virtual connection. A valid identification is a non-negative number.

**Returns:**

- ◇ **IX\_SUCCESS** : The VC has been successfully established on this port. The client may begin to submit demand on this VC.
- ◇ **IX\_ATMSCH\_RET\_NOT\_ADMITTED** : The VC cannot be established on this port because there is insufficient upstream capacity available to support the requested traffic contract descriptor
- ◇ **IX\_FAIL** : Input data are invalid. VC has not been established.

```
ixAtmSchVcQueueClear ( IxAtmLogicalPort    port,
                       IxAtmSchedulerVcId vcId
                       )
```

The client calls this function to remove all currently queued cells from a registered VC. The pending cell count for the specified VC is reset to zero.

This interface is structurally compatible with the `IxAtmdAccSchQueueClear` callback type definition required for IXP425 ATM scheduler interoperability.

**Parameters:**

<i>IxAtmLogicalPort</i> <i>port</i>	Specifies the ATM port on which the VC to be cleared is established.
<i>IxAtmSchedulerVcId</i> <i>vcId</i>	Identifies the VC to be cleared. This is the value returned by the <b>ixAtmSchVcModelSetup</b> call which established the relevant VC.

**Returns:**

- ◇ **IX\_SUCCESS** : The VC queue has been successfully cleared.
- ◇ **IX\_FAIL** : The input are invalid. No VC queue is modified.

**Warning:**

`IxAtmSch` assumes that the calling software ensures that calls to `ixAtmSchVcQueueUpdate`, `ixAtmSchVcQueueClear` and `ixAtmSchTableUpdate` are both self and mutually exclusive for the same port.

**See also:**

`ixAtmSchVcQueueUpdate()`, `ixAtmSchVcQueueClear()`, `ixAtmSchTableUpdate()`.

```

ixAtmSchVcQueueUpdate ( IxAtmLogicalPort    port,
                        IxAtmSchedulerVcId  vcId,
                        unsigned int          numberOfCells
                        )

```

The client calls this function to notify IxAtmSch that the user of a VC has submitted cells for transmission.

This information is stored, aggregated from a number of calls to ixAtmSchVcQueueUpdate and eventually used in the call to ixAtmSchTableUpdate.

Normally IxAtmSch will update the VC queue by adding the number of cells to the current queue length. However, if IxAtmSch determines that the user has over-submitted for the VC and exceeded its transmission quota the queue request can be rejected. The user should resubmit the request later when the queue has been depleted.

This implementation of ixAtmSchVcQueueUpdate uses no operating system or external facilities, either directly or indirectly. This allows clients to call this function from within an interrupt handler.

This interface is structurally compatible with the IxAtmAccSchQueueUpdate callback type definition required for IXP425 ATM scheduler interoperability.

**Parameters:**

<i>IxAtmLogicalPort port</i>	Specifies the ATM port on which the VC to be updated is established.
<i>IxAtmSchedulerVcId vcId</i>	Identifies the VC to be updated. This is the value returned by the <b>ixAtmSchVcModelSetup</b> call which established the relevant VC.
<i>unsigned int numberOfCells</i>	Indicates how many ATM cells should be added to the queue for this VC.

**Returns:**

- ◇ **IX\_SUCCESS** : The VC queue has been successfully updated.
- ◇ **IX\_ATMSCH\_RET\_QUEUE\_FULL** : The VC queue has reached a preset limit. This indicates the client has over-submitted and exceeded its transmission quota. The request is rejected. The VC queue is not updated. The VC user is advised to resubmit the request later.
- ◇ **IX\_FAIL** : The input are invalid. No VC queue is updated.

**Warning:**

IxAtmSch assumes that the calling software ensures that calls to ixAtmSchVcQueueUpdate, ixAtmSchVcQueueClear and ixAtmSchTableUpdate are both self and mutually exclusive for the same port.

**See also:**

**ixAtmSchVcQueueUpdate(), ixAtmSchVcQueueClear(), ixAtmSchTableUpdate().**

# IXP425 ATM Types (IxAtmTypes)

The common set of types used in many Atm components.

## Data Structures

struct **IxAtmScheduleTable**

*This structure defines a schedule table which gives details on which data (from which VCs) should be transmitted for a forthcoming period of time for a particular port and the order in which that data should be transmitted.*

struct **IxAtmScheduleTableEntry**

*ATM Schedule Table entry.*

struct **IxAtmTrafficDescriptor**

*Structure describing an ATM traffic contract for a Virtual Connection (VC).*

## Defines

#define **IX\_ATM\_CELL\_PAYLOAD\_SIZE**

*Size of a ATM cell payload.*

#define **IX\_ATM\_CELL\_SIZE**

*Size of a ATM cell, including header.*

#define **IX\_ATM\_CELL\_SIZE\_NO\_HEC**

*Size of a ATM cell, excluding HEC byte.*

#define **IX\_ATM\_OAM\_CELL\_SIZE\_NO\_HEC**

*Size of a OAM cell, excluding HEC byte.*

#define **IX\_ATM\_AAL0\_48\_CELL\_PAYLOAD\_SIZE**

*Size of a AAL0 48 Cell payload.*

#define **IX\_ATM\_AAL5\_CELL\_PAYLOAD\_SIZE**

*Size of a AAL5 Cell payload.*

#define **IX\_ATM\_AAL0\_52\_CELL\_SIZE\_NO\_HEC**

*Size of a AAL0 52 Cell, excluding HEC byte.*

#define **IX\_ATM\_MAX\_VPI**

*Maximum value of an ATM VPI.*

#define **IX\_ATM\_MAX\_VCI**

*Maximum value of an ATM VCI.*

#define **IX\_ATM\_MAX\_NUM\_AAL\_VCS**

*Maximum number of active AAL5/AAL0 VCs in the system.*

**#define IX\_ATM\_MAX\_NUM\_VC**

*Maximum number of active AAL5/AAL0 VCs in the system The use of this macro is depreciated, it is retained for backward compatiblity. For current software release and beyond the define IX\_ATM\_MAX\_NUM\_AAL\_VC should be used.*

**#define IX\_ATM\_MAX\_NUM\_OAM\_TX\_VCS**

*Maximum number of active OAM Tx VCs in the system, 1 OAM VC per port.*

**#define IX\_ATM\_MAX\_NUM\_OAM\_RX\_VCS**

*Maximum number of active OAM Rx VCs in the system, 1 OAM VC shared accross all ports.*

**#define IX\_ATM\_MAX\_NUM\_AAL\_OAM\_TX\_VCS**

*Maximum number of active AAL5/AAL0/OAM Tx VCs in the system.*

**#define IX\_ATM\_MAX\_NUM\_AAL\_OAM\_RX\_VCS**

*Maximum number of active AAL5/AAL0/OAM Rx VCs in the system.*

**#define IX\_ATM\_IDLE\_CELLS\_CONNID**

*VC Id used to indicate idle cells in the returned schedule table.*

**#define IX\_ATM\_CELL\_HEADER\_VCI\_GET(cellHeader)**

*get the VCI field from a cell header*

**#define IX\_ATM\_CELL\_HEADER\_VPI\_GET(cellHeader)**

*get the VPI field from a cell header*

**#define IX\_ATM\_CELL\_HEADER\_PTI\_GET(cellHeader)**

*get the PTI field from a cell header*

## Typedefs

typedef

unsigned

int **IxAtmCellHeader**

*ATM Cell Header, does not contain 4 byte HEC, added by NPE-A.*

typedef

unsigned

int **IxAtmConnId**

*ATM VC data connection identifier.*

typedef int **IxAtmSchedulerVcId**

*ATM VC scheduling connection identifier.*

typedef

unsigned

int **IxAtmNpeRxVcId**

*ATM Rx VC identifier used by the ATM Npe.*

## Enumerations

```
enum IxAtmLogicalPort {  
    IX_UTOPIA_PORT_0,  
    IX_UTOPIA_PORT_1,  
    IX_UTOPIA_PORT_2,  
    IX_UTOPIA_PORT_3,  
    IX_UTOPIA_PORT_4,  
    IX_UTOPIA_PORT_5,  
    IX_UTOPIA_PORT_6,  
    IX_UTOPIA_PORT_7,  
    IX_UTOPIA_PORT_8,  
    IX_UTOPIA_PORT_9,  
    IX_UTOPIA_PORT_10,  
    IX_UTOPIA_PORT_11,  
    IX_UTOPIA_MAX_PORTS  
}
```

*Logical Port Definitions :.*

```
enum IxAtmServiceCategory {  
    IX_ATM_CBR,  
    IX_ATM_RTVBR,  
    IX_ATM_VBR,  
    IX_ATM_UBR,  
    IX_ATM_ABR  
}
```

*Enumerated type representing available ATM service categories. For more informatoin on these categories, see "Traffic Management Specification" v4.1, published by the ATM Forum – <http://www.atmforum.com>.*

```
enum IxAtmRxQueueId {  
    IX_ATM_RX_A,  
    IX_ATM_RX_B,  
    IX_ATM_MAX_RX_STREAMS  
}
```

*Rx Queue Type for RX traffic.*

---

## Detailed Description

The common set of types used in many Atm components.

---

# Define Documentation

```
#define IX_ATM_AAL0_48_CELL_PAYLOAD_SIZE
```

Size of a AAL0 48 Cell payload.

Definition at line **121** of file **IxAtmTypes.h**.

```
#define IX_ATM_AAL0_52_CELL_SIZE_NO_HEC
```

Size of a AAL0 52 Cell, excluding HEC byte.

Definition at line **133** of file **IxAtmTypes.h**.

```
#define IX_ATM_AAL5_CELL_PAYLOAD_SIZE
```

Size of a AAL5 Cell payload.

Definition at line **127** of file **IxAtmTypes.h**.

```
#define IX_ATM_CELL_HEADER_PTI_GET ( cellHeader )
```

get the PTI field from a cell header

Definition at line **216** of file **IxAtmTypes.h**.

```
#define IX_ATM_CELL_HEADER_VCI_GET ( cellHeader )
```

get the VCI field from a cell header

Definition at line **202** of file **IxAtmTypes.h**.

```
#define IX_ATM_CELL_HEADER_VPI_GET ( cellHeader )
```

get the VPI field from a cell header

Definition at line **209** of file **IxAtmTypes.h**.

```
#define IX_ATM_CELL_PAYLOAD_SIZE
```

Size of a ATM cell payload.



Definition at line **97** of file **IxAtmTypes.h**.

```
#define IX_ATM_CELL_SIZE
```

Size of a ATM cell, including header.

Definition at line **103** of file **IxAtmTypes.h**.

```
#define IX_ATM_CELL_SIZE_NO_HEC
```

Size of a ATM cell, excluding HEC byte.

Definition at line **109** of file **IxAtmTypes.h**.

```
#define IX_ATM_IDLE_CELLS_CONNID
```

VC Id used to indicate idle cells in the returned schedule table.

Definition at line **195** of file **IxAtmTypes.h**.

```
#define IX_ATM_MAX_NUM_AAL_OAM_RX_VCS
```

Maximum number of active AAL5/AAL0/OAM Rx VCs in the system.

Definition at line **189** of file **IxAtmTypes.h**.

```
#define IX_ATM_MAX_NUM_AAL_OAM_TX_VCS
```

Maximum number of active AAL5/AAL0/OAM Tx VCs in the system.

Definition at line **183** of file **IxAtmTypes.h**.

```
#define IX_ATM_MAX_NUM_AAL_VCS
```

Maximum number of active AAL5/AAL0 VCs in the system.

Definition at line **152** of file **IxAtmTypes.h**.

```
#define IX_ATM_MAX_NUM_OAM_RX_VCS
```

Maximum number of active OAM Rx VCs in the system, 1 OAM VC shared accross all ports.

Definition at line **177** of file **IxAtmTypes.h**.

```
#define IX_ATM_MAX_NUM_OAM_TX_VCS
```

Maximum number of active OAM Tx VCs in the system, 1 OAM VC per port.

Definition at line **170** of file **IxAtmTypes.h**.

```
#define IX_ATM_MAX_NUM_VC
```

Maximum number of active AAL5/AAL0 VCs in the system The use of this macro is depreciated, it is retained for backward compatiblity. For current software release and beyond the define IX\_ATM\_MAX\_NUM\_AAL\_VC should be used.

Definition at line **161** of file **IxAtmTypes.h**.

```
#define IX_ATM_MAX_VCI
```

Maximum value of an ATM VCI.

Definition at line **146** of file **IxAtmTypes.h**.

```
#define IX_ATM_MAX_VPI
```

Maximum value of an ATM VPI.

Definition at line **140** of file **IxAtmTypes.h**.

```
#define IX_ATM_OAM_CELL_SIZE_NO_HEC
```

Size of a OAM cell, excluding HEC byte.

Definition at line **115** of file **IxAtmTypes.h**.

---

## Typedef Documentation

```
IxAtmCellHeader
```

ATM Cell Header, does not contain 4 byte HEC, added by NPE–A.

Definition at line **224** of file **IxAtmTypes.h**.

## IxAtmConnId

ATM VC data connection identifier.

This is generated by IxAtmdAcc when a successful connection is made on a VC. The is the ID by which IxAtmdAcc knows an active VC and should be used in IxAtmdAcc API calls to reference a specific VC.

Definition at line **312** of file **IxAtmTypes.h**.

## IxAtmNpeRxVcId

ATM Rx VC identifier used by the ATM Npe.

This Id is generated by IxAtmdAcc when a successful data connection is made on a rx VC.

Definition at line **336** of file **IxAtmTypes.h**.

## IxAtmSchedulerVcId

ATM VC scheduling connection identifier.

This id is generated and used by ATM Tx controller, generally the traffic shaper (e.g. IxAtmSch). The IxAtmdAcc component will request one of these Ids whenever a data connection on a Tx VC is requested. This ID will be used in callbacks to the ATM Transmission Ctrl s/w (e.g. IxAtmm) to reference a particular VC.

Definition at line **326** of file **IxAtmTypes.h**.

---

# Enumeration Type Documentation

## enum IxAtmLogicalPort

Logical Port Definitions :.

Only 1 port is available in SPHY configuration 12 ports are enabled in MPHY configuration

### ***Enumeration values:***

<i>IX_UTOPIA_PORT_0</i>	Port 0.
<i>IX_UTOPIA_PORT_1</i>	Port 1.
<i>IX_UTOPIA_PORT_2</i>	Port 2.
<i>IX_UTOPIA_PORT_3</i>	Port 3.
<i>IX_UTOPIA_PORT_4</i>	Port 4.
<i>IX_UTOPIA_PORT_5</i>	Port 5.
<i>IX_UTOPIA_PORT_6</i>	Port 6.
<i>IX_UTOPIA_PORT_7</i>	Port 7.

<i>IX_UTOPIA_PORT_8</i>	Port 8.
<i>IX_UTOPIA_PORT_9</i>	Port 9.
<i>IX_UTOPIA_PORT_10</i>	Port 10.
<i>IX_UTOPIA_PORT_11</i>	Port 11.
<i>IX_UTOPIA_MAX_PORTS</i>	Not a port – just a definition for the maximum possible ports.

Definition at line **72** of file **IxAtmTypes.h**.

```
enum IxAtmRxQueueId
```

Rx Queue Type for RX traffic.

IxAtmRxQueueId defines the queues involved for receiving data.

There are two queues to facilitate prioritisation handling and processing the 2 queues with different algorithms and constraints

e.g. : one queue can carry voice (or time–critical traffic), the other queue can carry non–voice traffic

***Enumeration values:***

<i>IX_ATM_RX_A</i>	RX queue A.
<i>IX_ATM_RX_B</i>	RX queue B.
<i>IX_ATM_MAX_RX_STREAMS</i>	Maximum number of RX streams.

Definition at line **261** of file **IxAtmTypes.h**.

```
enum IxAtmServiceCategory
```

Enumerated type representing available ATM service categories. For more informatoin on these categories, see "Traffic Management Specification" v4.1, published by the ATM Forum – <http://www.atmforum.com>.

***Enumeration values:***

<i>IX_ATM_CBR</i>	Constant Bit Rate.
<i>IX_ATM_RTVBR</i>	Real Time Variable Bit Rate.
<i>IX_ATM_VBR</i>	Variable Bit Rate.
<i>IX_ATM_UBR</i>	Unspecified Bit Rate.
<i>IX_ATM_ABR</i>	Available Bit Rate (not supported).

Definition at line **235** of file **IxAtmTypes.h**.

# IXP425 DMA Types (IxDmaTypes)

The common set of types used in the DMA component.

## Enumerations

```
enum IxDmaReturnStatus {  
    IX_DMA_SUCCESS,  
    IX_DMA_FAIL,  
    IX_DMA_INVALID_TRANSFER_WIDTH,  
    IX_DMA_INVALID_TRANSFER_LENGTH,  
    IX_DMA_INVALID_TRANSFER_MODE,  
    IX_DMA_INVALID_ADDRESS_MODE,  
    IX_DMA_REQUEST_FIFO_FULL  
}
```

*Dma return status definitions.*

```
enum IxDmaTransferMode {  
    IX_DMA_COPY_CLEAR,  
    IX_DMA_COPY,  
    IX_DMA_COPY_BYTE_SWAP,  
    IX_DMA_COPY_REVERSE,  
    IX_DMA_TRANSFER_MODE_INVALID  
}
```

*Dma transfer mode definitions.*

```
enum IxDmaAddressingMode {  
    IX_DMA_INC_SRC_INC_DST,  
    IX_DMA_INC_SRC_FIX_DST,  
    IX_DMA_FIX_SRC_INC_DST,  
    IX_DMA_FIX_SRC_FIX_DST,  
    IX_DMA_ADDRESSING_MODE_INVALID  
}
```

*Dma addressing mode definitions.*

```
enum IxDmaTransferWidth {  
    IX_DMA_32_SRC_32_DST,  
    IX_DMA_32_SRC_16_DST,  
    IX_DMA_32_SRC_8_DST,  
    IX_DMA_16_SRC_32_DST,  
    IX_DMA_16_SRC_16_DST,  
    IX_DMA_16_SRC_8_DST,  
    IX_DMA_8_SRC_32_DST,  
    IX_DMA_8_SRC_16_DST,  
    IX_DMA_8_SRC_8_DST,  
    IX_DMA_8_SRC_BURST_DST,  
    IX_DMA_16_SRC_BURST_DST,  
    IX_DMA_32_SRC_BURST_DST,  
}
```

```

IX_DMA_BURST_SRC_8_DST,
IX_DMA_BURST_SRC_16_DST,
IX_DMA_BURST_SRC_32_DST,
IX_DMA_BURST_SRC_BURST_DST,
IX_DMA_TRANSFER_WIDTH_INVALID
}

```

*Dma transfer width definitions Fixed addresses (either source or destination) do not support burst transfer width.*

```

enum IxDmaNpeId {
    IX_DMA_NPEID_NPEA,
    IX_DMA_NPEID_NPEB,
    IX_DMA_NPEID_NPEC,
    IX_DMA_NPEID_MAX
}

```

*NpeId numbers to identify NPE A, B or C.*

---

## Detailed Description

The common set of types used in the DMA component.

---

## Enumeration Type Documentation

```
enum IxDmaAddressingMode
```

Dma addressing mode definitions.

**Note:**

Fixed source address to fixed destination address addressing mode is not supported.

**Enumeration values:**

<i>IX_DMA_INC_SRC_INC_DST</i>	Incremental source address to incremental destination address.
<i>IX_DMA_INC_SRC_FIX_DST</i>	Incremental source address to incremental destination address.
<i>IX_DMA_FIX_SRC_INC_DST</i>	Incremental source address to incremental destination address.
<i>IX_DMA_FIX_SRC_FIX_DST</i>	Incremental source address to incremental destination address.
<i>IX_DMA_ADDRESSING_MODE_INVALID</i>	Invalid Addressing Mode.

Definition at line **107** of file **IxDmaAcc.h**.

```
enum IxDmaNpeId
```

NpeId numbers to identify NPE A, B or C.

**Enumeration values:**

*IX\_DMA\_NPEID\_NPEA* Identifies NPE  
A.  
*IX\_DMA\_NPEID\_NPEB* Identifies NPE  
B.  
*IX\_DMA\_NPEID\_NPEC* Identifies NPE  
C.  
*IX\_DMA\_NPEID\_MAX* Total Number  
of NPEs.

Definition at line **149** of file **IxDmaAcc.h**.

enum IxDmaReturnStatus

Dma return status definitions.

**Enumeration values:**

<i>IX_DMA_SUCCESS</i>	DMA Transfer Success.
<i>IX_DMA_FAIL</i>	DMA Transfer Fail.
<i>IX_DMA_INVALID_TRANSFER_WIDTH</i>	Invalid transfer width.
<i>IX_DMA_INVALID_TRANSFER_LENGTH</i>	Invalid transfer length.
<i>IX_DMA_INVALID_TRANSFER_MODE</i>	Invalid transfer mode.
<i>IX_DMA_INVALID_ADDRESS_MODE</i>	Invalid address mode.
<i>IX_DMA_REQUEST_FIFO_FULL</i>	DMA request queue is full.

Definition at line **75** of file **IxDmaAcc.h**.

enum IxDmaTransferMode

Dma transfer mode definitions.

**Note:**

Copy and byte swap, and copy and reverse modes only support multiples of word data length.

**Enumeration values:**

<i>IX_DMA_COPY_CLEAR</i>	copy and clear source
<i>IX_DMA_COPY</i>	copy

<i>IX_DMA_COPY_BYTE_SWAP</i>	copy and byte swap (endian)
<i>IX_DMA_COPY_REVERSE</i>	copy and reverse
<i>IX_DMA_TRANSFER_MODE_INVALID</i>	Invalid transfer mode.

Definition at line **92** of file **IxDmaAcc.h**.

#### enum IxDmaTransferWidth

Dma transfer width definitions Fixed addresses (either source or destination) do not support burst transfer width.

##### **Enumeration values:**

<i>IX_DMA_32_SRC_32_DST</i>	32-bit src to 32-bit dst
<i>IX_DMA_32_SRC_16_DST</i>	32-bit src to 16-bit dst
<i>IX_DMA_32_SRC_8_DST</i>	32-bit src to 8-bit dst
<i>IX_DMA_16_SRC_32_DST</i>	16-bit src to 32-bit dst
<i>IX_DMA_16_SRC_16_DST</i>	16-bit src to 16-bit dst
<i>IX_DMA_16_SRC_8_DST</i>	16-bit src to 8-bit dst
<i>IX_DMA_8_SRC_32_DST</i>	8-bit src to 32-bit dst
<i>IX_DMA_8_SRC_16_DST</i>	8-bit src to 16-bit dst
<i>IX_DMA_8_SRC_8_DST</i>	8-bit src to 8-bit dst
<i>IX_DMA_8_SRC_BURST_DST</i>	8-bit src to burst dst – Not supported for fixed destination address
<i>IX_DMA_16_SRC_BURST_DST</i>	16-bit src to burst dst – Not supported for fixed destination address
<i>IX_DMA_32_SRC_BURST_DST</i>	32-bit src to burst dst – Not supported for fixed destination address
<i>IX_DMA_BURST_SRC_8_DST</i>	burst src to 8-bit dst – Not supported for fixed source address
<i>IX_DMA_BURST_SRC_16_DST</i>	burst src to 16-bit dst – Not supported for fixed source address
<i>IX_DMA_BURST_SRC_32_DST</i>	burst src to 32-bit dst – Not supported for fixed source address
<i>IX_DMA_BURST_SRC_BURST_DST</i>	burst src to burst dst – Not supported for fixed source and destination address
<i>IX_DMA_TRANSFER_WIDTH_INVALID</i>	Invalid transfer width.

Definition at line **122** of file **IxDmaAcc.h**.



# IXP425 DMA Access Driver (IxDmaAcc) API

The public API for the IXP425 IxDmaAcc component.

## Defines

```
#define IX_DMA_REQUEST_FULL  
DMA request queue is full This constant is a return value used to  
tell the user that the IxDmaAcc queue is full.
```

## Typedefs

```
typedef UINT32 IxDmaAccRequestId  
DMA Request Id type.  
  
typedef void(* IxDmaAccDmaCompleteCallback)(IxDmaReturnStatus status)  
DMA completion notification This function is called to notify a  
client that the DMA has been completed.
```

## Functions

```
PUBLIC IX_STATUS ixDmaAccInit (INpeId npeId)  
Initialise the DMA Access component This function will initialise  
the DMA Access component internals.  
  
PUBLIC IxDmaReturnStatus ixDmaAccDmaTransfer (IxDmaAccDmaCompleteCallback  
callback, UINT32 SourceAddr, UINT32 DestinationAddr, UINT16  
TransferLength, IxDmaTransferMode TransferMode,  
IxDmaAddressingMode AddressingMode,  
IxDmaTransferWidth TransferWidth)  
Perform DMA transfer This function will perform DMA transfer  
between devices within the IXP425 memory map.  
  
PUBLIC IX_STATUS ixDmaAccShow (void)  
Display some component information for debug purposes Show  
some internal operation information relating to the DMA service.  
At a minimum the following will show. –the number of the DMA  
pend (in queue).
```

---

## Detailed Description

The public API for the IXP425 IxDmaAcc component.

---

## Define Documentation

```
#define IX_DMA_REQUEST_FULL
```

DMA request queue is full This constant is a return value used to tell the user that the IxDmaAcc queue is full.

Definition at line **179** of file **IxDmaAcc.h**.

---

## Typedef Documentation

```
typedef void(* IxDmaAccDmaCompleteCallback)(IxDmaReturnStatus status)
```

DMA completion notification This function is called to notify a client that the DMA has been completed.

**Parameters:**

*IxDmaReturnStatus* Status reporting to client

Definition at line **188** of file **IxDmaAcc.h**.

```
typedef UINT32 IxDmaAccRequestId
```

DMA Request Id type.

Definition at line **169** of file **IxDmaAcc.h**.

---

## Function Documentation

```
ixDmaAccDmaTransfer ( IxDmaAccDmaCompleteCallback callback,  
                     UINT32 SourceAddr,  
                     UINT32 DestinationAddr,  
                     UINT16 TransferLength,  
                     IxDmaTransferMode TransferMode,  
                     IxDmaAddressingMode AddressingMode,  
                     IxDmaTransferWidth TransferWidth  
                     )
```

Perform DMA transfer This function will perform DMA transfer between devices within the IXP425 memory map.

**Note:**

The following are restrictions for IxDmaAccDmaTransfer:

- ◇ The function is non re-entrant.
- ◇ The function assumes host devices are operating in big-endian mode.
- ◇ Fixed address does not support burst transfer width
- ◇ Fixed source address to fixed destination address mode is not supported
- ◇ The incrementing source address for expansion bus will not support a burst transfer width and copy and clear mode

**Parameters:**

<i>IxDmaAccDmaCompleteCallback</i> <i>callback</i>	function pointer to be stored and called when the DMA transfer is completed. This cannot be NULL.
<i>ixDmaSourceAddr</i>	Starting address of DMA source. Must be a valid IXP425 memory map address.
<i>ixDmaDestinationAddr</i>	Starting address of DMA destination. Must be a valid IXP425 memory map address.
<i>ixDmaTransferLength</i>	The size of DMA data transfer. The range must be from 1–64Kbyte
<i>ixDmaTransferMode</i>	The DMA transfer mode
<i>ixDmaAddressingMode</i>	The DMA addressing mode
<i>ixTransferWidth</i>	The DMA transfer width

**Returns:**

- ◇ IX\_DMA\_SUCCESS Notification that the DMA request is succesful
- ◇ IX\_DMA\_FAIL IxDmaAcc not yet initialised or some internal error has occurred
- ◇ IX\_DMA\_INVALID\_TRANSFER\_WIDTH Transfer width is nit valid
- ◇ IX\_DMA\_INVALID\_TRANSFER\_LENGTH Transfer length outside of valid range
- ◇ IX\_DMA\_INVALID\_TRANSFER\_MODE Transfer Mode not valid
- ◇ IX\_DMA\_REQUEST\_FIFO\_FULL IxDmaAcc request queue is full

```
ixDmaAccInit ( IXNpeDlNpeId npeId )
```

Initialise the DMA Access component This function will initialise the DMA Access component internals.

**Parameters:**

*npeId* – NPE to use for Dma Transfer

**Returns:**

- ◇ IX\_SUCCESS succesfully initialised the component

◇ IX\_FAIL Initialisation failed for some unspecified internal reason.

`ixDmaAccShow ( void )`

Display some component information for debug purposes Show some internal operation information relating to the DMA service. At a minimum the following will show. –the number of the DMA pend (in queue).

***Parameters:***

*None*

***Returns:***

◇ None

# IXP425 Ethernet Access (IxEthAcc) API

ethAcc is a library that does provides access to the internal IXP425 10/100Bt Ethernet MACs.

## Data Structures

struct **IxEthAccMacAddr**

*The IEEE 802.3 Ethernet MAC address structure.*

struct **IxEthEthObjStats**

*This struct defines the statistics returned by this component. The component returns MIB2 EthObj variables which should are obtained from the hardware or maintained by this component.*

## Defines

#define **IX\_ETH\_ACC\_NUMBER\_OF\_PORTS**

*Defines related to the number of NPE's and mapping between PortId and NPE.*

#define **IX\_IEEE803\_MAC\_ADDRESS\_SIZE**

*Defines the size of the MAC address NPE.*

#define **IX\_ETH\_ACC\_NUM\_TX\_PRIORITIES**

*The number of transmit priorities.*

#define **IX\_ETHACC\_RX\_MBUF\_MIN\_SIZE**

*This defines the recommended minimum size of MBUF's submitted to the frame receive service.*

#define **IXP425\_ETH\_ACC\_MII\_MAX\_ADDR**

*This defines the highest MII address of any attached PHYs.*

#define **ixEthAccMiiPhyScan**(phyPresent)

*: deprecated API entry point. This definition ensures backward compatibility*

#define **ixEthAccMiiPhyConfig**(phyAddr, speed100, fullDuplex, autonegotiate)

*: deprecated API entry point. This definition ensures backward compatibility*

#define **ixEthAccMiiPhyReset**(phyAddr)

*: deprecated API entry point. This definition ensures backward compatibility*

#define **ixEthAccMiiLinkStatus**(phyAddr, linkUp, speed100, fullDuplex, autoneg)

*: deprecated API entry point. This definition ensures backward compatibility*

#define **ixEthAccMiiShow**(phyAddr)

*: deprecated API entry point. This definition ensures backward compatibility*

## Typedefs

typedef void(\* **IxEthAccPortTxDoneCallback** )(UINT32 callbackTag, IX\_MBUF \*buffer)  
*Function prototype for Ethernet Tx Buffer Done callback. Registered via ixEthAccTxBufferDoneCallbackRegister.*

typedef void(\* **IxEthAccPortRxCallback** )(UINT32 callbackTag, IX\_MBUF \*buffer, **IxEthAccPortId** portId)  
*Function prototype for Ethernet Frame Rx callback. Registered via ixEthAccPortRxCallbackRegister.*

## Enumerations

enum **IxEthAccStatus** {  
    **IX\_ETH\_ACC\_SUCCESS**,  
    **IX\_ETH\_ACC\_FAIL**,  
    **IX\_ETH\_ACC\_INVALID\_PORT**,  
    **IX\_ETH\_ACC\_PORT\_UNINITIALIZED**,  
    **IX\_ETH\_ACC\_MAC\_UNINITIALIZED**,  
    **IX\_ETH\_ACC\_INVALID\_ARG**,  
    **IX\_ETH\_TX\_Q\_FULL**,  
    **IX\_ETH\_ACC\_NO\_SUCH\_ADDR**  
}  
*This is an enum to define the Ethernet Access status.*

enum **IxEthAccPortId** {  
    **IX\_ETH\_PORT\_1**,  
    **IX\_ETH\_PORT\_2**  
}  
*This is an enum to define the IXP425 Mac Ethernet device.*

enum **IxEthAccTxPriority** {  
    **IX\_ETH\_ACC\_TX\_PRIORITY\_0**,  
    **IX\_ETH\_ACC\_TX\_PRIORITY\_1**,  
    **IX\_ETH\_ACC\_TX\_PRIORITY\_2**,  
    **IX\_ETH\_ACC\_TX\_PRIORITY\_3**,  
    **IX\_ETH\_ACC\_TX\_PRIORITY\_4**,  
    **IX\_ETH\_ACC\_TX\_PRIORITY\_5**,  
    **IX\_ETH\_ACC\_TX\_PRIORITY\_6**,  
    **IX\_ETH\_ACC\_TX\_PRIORITY\_7**,  
    **IX\_ETH\_ACC\_TX\_DEFAULT\_PRIORITY**  
}  
*enum to submit a frame with relative priority.*

enum **IxEthAccDuplexMode** {  
    **IX\_ETH\_ACC\_FULL\_DUPLEX**,  
    **IX\_ETH\_ACC\_HALF\_DUPLEX**  
}

*Definition to provision the duplex mode of the MAC.*

```
enum IxEthAccTxSchedulerDiscipline {  
    FIFO_NO_PRIORITY,  
    FIFO_PRIORITY  
}
```

*Definition for the port transmit scheduling discipline Definition for the port transmit scheduling discipline.*

## Functions

**IxEthAccStatus** **ixEthAccInit** (void)  
*Initialize the Ethernet Access Service.*

void **ixEthAccUnload** (void)  
*Unload the Ethernet Access Service.*

**IxEthAccStatus** **ixEthAccPortInit** (**IxEthAccPortId** portId)  
*Initialize an Ethernet MAC Port.*

**IxEthAccStatus** **ixEthAccPortTxFrameSubmit** (**IxEthAccPortId** portId, IX\_MBUF \*buffer,  
**IxEthAccTxPriority** priority)  
*This function shall be used to submit MBUFs buffers for transmission on a particular MAC device.*

**IxEthAccStatus** **ixEthAccPortTxDoneCallbackRegister** (**IxEthAccPortId** portId,  
**IxEthAccPortTxDoneCallback** txCallbackFn, UINT32 callbackTag)  
*This function registers a callback function to facilitate the return of transmit buffers to the user.*

**IxEthAccStatus** **ixEthAccPortRxCallbackRegister** (**IxEthAccPortId** portId, **IxEthAccPortRxCallback**  
rxCallbackFn, UINT32 callbackTag)  
*The function registered through this function shall be called once per received Ethernet frame.*

**IxEthAccStatus** **ixEthAccPortRxFreeReplenish** (**IxEthAccPortId** portId, IX\_MBUF \*buffer)  
*This function provides buffers for the Ethernet receive path.*

**IxEthAccStatus** **ixEthAccPortEnable** (**IxEthAccPortId** portId)  
*Enable a port.*

**IxEthAccStatus** **ixEthAccPortDisable** (**IxEthAccPortId** portId)  
*Disable a port.*

**IxEthAccStatus** **ixEthAccPortEnabledQuery** (**IxEthAccPortId** portId, BOOL \*enabled)  
*Get the enabled state of a port.*

**IxEthAccStatus** **ixEthAccPortPromiscuousModeClear** (**IxEthAccPortId** portId)

*Put the Ethernet MAC device in non-promiscuous mode.*

**IxEthAccStatus ixEthAccPortPromiscuousModeSet (IxEthAccPortId portId)**

*Put the MAC device in promiscuous mode.*

**IxEthAccStatus ixEthAccPortUnicastMacAddressSet (IxEthAccPortId portId, IxEthAccMacAddr \*macAddr)**

*Configure unicast MAC address for a particular port.*

**IxEthAccStatus ixEthAccPortUnicastMacAddressGet (IxEthAccPortId portId, IxEthAccMacAddr \*macAddr)**

*Get unicast MAC address for a particular MAC port.*

**IxEthAccStatus ixEthAccPortMulticastAddressJoin (IxEthAccPortId portId, IxEthAccMacAddr \*macAddr)**

*ADD a multicast address to the MAC address table.*

**IxEthAccStatus ixEthAccPortMulticastAddressJoinAll (IxEthAccPortId portId)**

*Filter all frames with multicast dest.*

**IxEthAccStatus ixEthAccPortMulticastAddressLeave (IxEthAccPortId portId, IxEthAccMacAddr \*macAddr)**

*Remove a multicast address from the MAC address table.*

**IxEthAccStatus ixEthAccPortMulticastAddressLeaveAll (IxEthAccPortId portId)**

*Clear the MAC address table.*

**IxEthAccStatus ixEthAccPortUnicastAddressShow (IxEthAccPortId portId)**

*Display unicast address has been configured using ixEthAccUnicastMacAddressSet.*

**void ixEthAccPortMulticastAddressShow (IxEthAccPortId portId)**

*Display multicast address which have been configured using ixEthAccMulticastAddressJoin*

*Display multicast address which have been configured using ixEthAccMulticastAddressJoin.*

**IxEthAccStatus ixEthAccPortDuplexModeSet (IxEthAccPortId portId, IxEthAccDuplexMode mode)**

*Set the duplex mode for the MAC.*

**IxEthAccStatus ixEthAccPortDuplexModeGet (IxEthAccPortId portId, IxEthAccDuplexMode \*mode)**

*Get the duplex mode for the MAC.*

**IxEthAccStatus ixEthAccPortTxFrameAppendPaddingEnable (IxEthAccPortId portId)**

*Enable the appending of padding bytes to runt frames submitted to this port.*

**IxEthAccStatus ixEthAccPortTxFrameAppendPaddingDisable (IxEthAccPortId portId)**

*Disable the appending of padding bytes to the runt frames submitted to this port.*

**IxEthAccStatus ixEthAccPortTxFrameAppendFCSEnable (IxEthAccPortId portId)**

*Enable the appending of Ethernet FCS to all frames submitted to this port.*

**IxEthAccStatus ixEthAccPortTxFrameAppendFCSDisable (IxEthAccPortId portId)**



*Disable the appending of Ethernet FCS to all frames submitted to this port.*

**IxEthAccStatus ixEthAccPortRxFrameAppendFCSEnable (IxEthAccPortId portId)**

*Forward frames with FCS included in the receive buffer to the user.*

**IxEthAccStatus ixEthAccPortRxFrameAppendFCSDisable (IxEthAccPortId portId)**

*Disable the appending of Ethernet FCS to all frames submitted to this port.*

**IxEthAccStatus ixEthAccTxSchedulingDisciplineSet (IxEthAccPortId portId,  
IxEthAccTxSchedulerDiscipline sched)**

*Set the port scheduling to one of IxEthAccTxSchedulerDiscipline Set the port scheduling to one of IxEthAccTxSchedulerDiscipline.*

**IxEthAccStatus ixEthAccMibIIStatsGet (IxEthAccPortId portId, IxEthEthObjStats \*retStats)**

*Return the statistics maintained for a port. Return the statistics maintained for a port.*

**IxEthAccStatus ixEthAccMibIIStatsGetClear (IxEthAccPortId portId, IxEthEthObjStats \*retStats)**

*Return and clear the statistics maintained for a port. Return and clear the statistics maintained for a port.*

**IxEthAccStatus ixEthAccMibIIStatsClear (IxEthAccPortId portId)**

*Clear the statistics maintained for a port. Clear the statistics maintained for a port.*

**IxEthAccStatus ixEthAccMacInit (IxEthAccPortId portId)**

*Initialize the ethernet MAC settings.*

**void ixEthAccStatsShow (IxEthAccPortId portId)**

*Display a ports statistics on the standard io console using printf. Display a ports statistics on the standard io console using printf.*

**IxEthAccStatus ixEthAccMiiReadRtn (UINT8 phyAddr, UINT8 phyReg, UINT16 \*value)**

*Read a 16 bit value from a PHY.*

**IxEthAccStatus ixEthAccMiiWriteRtn (UINT8 phyAddr, UINT8 phyReg, UINT16 value)**

*Write a 16 bit value to a PHY.*

**IxEthAccStatus ixEthAccMiiStatsShow (UINT32 phyAddr)**

*Display detailed information on a specified PHY.*

---

## Detailed Description

ethAcc is a library that does provides access to the internal IXP425 10/100Bt Ethernet MACs.

---

## Define Documentation

```
#define IX_ETH_ACC_NUM_TX_PRIORITIES
```

The number of transmit priorities.

Definition at line **133** of file **IxEthAcc.h**.

```
#define IX_ETH_ACC_NUMBER_OF_PORTS
```

Defines related to the number of NPE's and mapping between PortId and NPE.

Definition at line **102** of file **IxEthAcc.h**.

```
#define IX_ETHACC_RX_MBUF_MIN_SIZE
```

This defines the recommended minimum size of MBUF's submitted to the frame receive service.

Definition at line **176** of file **IxEthAcc.h**.

```
#define IX_IEEE803_MAC_ADDRESS_SIZE
```

Defines the size of the MAC address NPE.

Definition at line **112** of file **IxEthAcc.h**.

```
#define ixEthAccMiiLinkStatus ( phyAddr,  
                                linkUp,  
                                speed100,  
                                fullDuplex,  
                                autoneg    )
```

: deprecated API entry point. This definition ensures backward compatibility

See **ixEthMiiLinkStatus**

**Note:**

this feature is board specific

Definition at line **1710** of file **IxEthAcc.h**.

```
#define ixEthAccMiiPhyConfig ( phyAddr,  
                                speed100,  
                                fullDuplex,  
                                autonegotiate )
```

: deprecated API entry point. This definition ensures backward compatibility

See **ixEthMiiPhyConfig**

**Note:**

this feature is board specific

Definition at line **1680** of file **IxEthAcc.h**.

```
#define ixEthAccMiiPhyReset ( phyAddr )
```

: deprecated API entry point. This definition ensures backward compatibility

See **ixEthMiiPhyReset**

**Note:**

this feature is board specific

Definition at line **1695** of file **IxEthAcc.h**.

```
#define ixEthAccMiiPhyScan ( phyPresent )
```

: deprecated API entry point. This definition ensures backward compatibility

See **ixEthMiiPhyScan**

**Note:**

this feature is board specific

Definition at line **1666** of file **IxEthAcc.h**.

```
#define ixEthAccMiiShow ( phyAddr )
```

: deprecated API entry point. This definition ensures backward compatibility

See **ixEthMiiPhyShow**

**Note:**

this feature is board specific

Definition at line **1727** of file **IxEthAcc.h**.

```
#define IXP425_ETH_ACC_MII_MAX_ADDR
```

This defines the highest MII address of any attached PHYs.

Max number for PHY address is 31, add on for range checking.

## Typedef Documentation

```
typedef void(* IxEthAccPortRxCallback)(UINT32 callbackTag, IX_MBUF *buffer, IxEthAccPortId portId)
```

Function prototype for Ethernet Frame Rx callback. Registered via *ixEthAccPortRxCallbackRegister*.

It is the responsibility of the user function to free any MBUF's which it receives.

- Reentrant – yes , The user provided function should be reentrant.
- ISR Callable – yes , The user provided function must be callable from an ISR.

### **Calling Context :**

This callback is called in the context of the queue manager dispatch loop *ixQmgrgrDispatcherLoopRun* within the **IXP425 Queue Manager (IxQMgr) API** component. The calling context may be from interrupt or high priority thread. The decision is system specific.

### **Parameters:**

- callbackTag* – This tag is that provided when the callback was registered for a particular MAC via *ixEthAccPortRxCallbackRegister*. It allows the same callback to be used for multiple MACs.
- mbuf* – Pointer to the Rx mbuf header. Mbufs may be chained if the frame length is greater than the supplied mbuf length.
- portId* – ID of the port which match the destination MAC address for this frame (The value is greater or equal to IX\_ETH\_DB\_NUMBER\_OF\_PORTS if the MAC address is not found in the copy of the database shared with the NPE)

### **Returns:**

void

### **Note:**

Buffers may not be filled up to the length supplied in *ixEthAccPortRxFreeReplenish()*. The firmware fills them to the previous 64 bytes boundary. The user has to be aware that the length of the received mbufs may be smaller than the length of the supplied mbufs. The mbuf header contains the following modified field

- IX\_MBUF\_PKT\_LEN is set in the header of the first mbuf and indicates the total frame size
- IX\_MBUF\_LEN is set each mbuf header and indicates the payload length
- IX\_MBUF\_NEXT\_BUFFER\_IN\_PKT\_PTR contains a pointer to the next mbuf, or NULL at the end of a chain.
- IX\_MBUF\_NEXT\_PKT\_IN\_CHAIN\_PTR is modified. Its value is reset to NULL
- IX\_MBUF\_FLAGS contains the bit 4 set for a broadcast packet and the bit 5 set for a multicast packet. Other bits are unmodified.

### **Warning:**

Any portID returned via the received callback is invalid if greater than 5. If your system defines more than 6 ports (0 to 5) in **IxEthDBPortDefs.h** you MUST check this value before using it in conjunction with the Ethernet Learning/Filtering Database. This is a constraint of an NPE Rx descriptor format limitation, which can report valid ports only between 0 and 5, and will use 6 for "reserved" and "7" for "not found", while the XScale

Ethernet Database can use these numbers for legitimate ports.

---

Definition at line **473** of file **IxEthAcc.h**.

```
typedef void(* IxEthAccPortTxDoneCallback)( UINT32 callbackTag, IX_MBUF *buffer )
```

Function prototype for Ethernet Tx Buffer Done callback. Registered via *ixEthAccTxBufferDoneCallbackRegister*.

This function is called once the previously submitted buffer is no longer required by this service. It may be returned upon successful transmission of the frame or shutdown of port prior to submission. The calling of this registered function is not a guarantee of successful transmission of the buffer.

- Reentrant – yes , The user provided function should be reentrant.
- ISR Callable – yes , The user provided function must be callable from an ISR.

**Calling Context :**

This callback is called in the context of the queue manager dispatch loop *ixQmgrgrDispatcherLoopRun* within the **IXP425 Queue Manager (IxQMgr) API** component. The calling context may be from interrupt or high priority thread. The decision is system specific.

**Parameters:**

*callbackTag* – This tag is that provided when the callback was registered for a particular MAC via *ixEthAccPortRxCallbackRegister*. It allows the same callback to be used for multiple MACs.

*mbuf* – Pointer to the Tx mbuf descriptor.

**Returns:**

void

**Note:**

The field IX\_MBUF\_NEXT\_PKT\_IN\_CHAIN\_PTR is modified by the access layer and reset to NULL.

---

Definition at line **368** of file **IxEthAcc.h**.

---

## Enumeration Type Documentation

```
enum IxEthAccDuplexMode
```

Definition to provision the duplex mode of the MAC.

**Enumeration values:**

*IX\_ETH\_ACC\_FULL\_DUPLEX* Full duplex operation of the MAC.

*IX\_ETH\_ACC\_HALF\_DUPLEX*

Half duplex operation of the  
MAC.

Definition at line **162** of file **IxEthAcc.h**.

enum IxEthAccPortId

This is an enum to define the IXP425 Mac Ethernet device.

**Enumeration values:**

*IX\_ETH\_PORT\_1* Ethernet Port 1.  
*IX\_ETH\_PORT\_2* Ethernet port 2.

Definition at line **88** of file **IxEthAcc.h**.

enum IxEthAccStatus

This is an enum to define the Ethernet Access status.

**Enumeration values:**

<i>IX_ETH_ACC_SUCCESS</i>	return success
<i>IX_ETH_ACC_FAIL</i>	return fail
<i>IX_ETH_ACC_INVALID_PORT</i>	return invalid port
<i>IX_ETH_ACC_PORT_UNINITIALIZED</i>	return uninitialized
<i>IX_ETH_ACC_MAC_UNINITIALIZED</i>	return MAC uninitialized
<i>IX_ETH_ACC_INVALID_ARG</i>	return invalid arg
<i>IX_ETH_TX_Q_FULL</i>	return tx queue is full
<i>IX_ETH_ACC_NO_SUCH_ADDR</i>	return no such address

Definition at line **71** of file **IxEthAcc.h**.

enum IxEthAccTxPriority

enum to submit a frame with relative priority.

**Enumeration values:**

<i>IX_ETH_ACC_TX_PRIORITY_0</i>	Lowest Priority submission.
<i>IX_ETH_ACC_TX_PRIORITY_1</i>	submission prority of 1 (0 is lowest)
<i>IX_ETH_ACC_TX_PRIORITY_2</i>	submission prority of 2 (0 is lowest)
<i>IX_ETH_ACC_TX_PRIORITY_3</i>	submission prority of 3 (0 is lowest)
<i>IX_ETH_ACC_TX_PRIORITY_4</i>	submission prority of 4 (0 is lowest)

<i>IX_ETH_ACC_TX_PRIORITY_5</i>	submission priority of 5 (0 is lowest)
<i>IX_ETH_ACC_TX_PRIORITY_6</i>	submission priority of 6 (0 is lowest)
<i>IX_ETH_ACC_TX_PRIORITY_7</i>	Highest priority submission.
<i>IX_ETH_ACC_TX_DEFAULT_PRIORITY</i>	By default send all pkts with lowest priority.

Definition at line **141** of file **IxEthAcc.h**.

```
enum IxEthAccTxSchedulerDiscipline
```

Definition for the port transmit scheduling discipline Definition for the port transmit scheduling discipline.

Select the port transmit scheduling discipline

- **FIFO : No Priority** : In this configuration all frames submitted to the access component shall be sub-mitted to the MAC hardware in the strict order in which it was received.
- **FIFO : Priority** : This shall be a very simple priority mechanism all submitted frames at a higher priority shall be forwarded to Ethernet MAC for transmission before lower priorities. There shall be no fairness mechanisms applied across different priorities. Higher priority frames could starve lower priority frames indefinitely.

**Enumeration values:**

*FIFO\_NO\_PRIORITY* frames submitted with no priority

*FIFO\_PRIORITY* higher priority frames submitted before lower priority

Definition at line **1333** of file **IxEthAcc.h**.

---

## Function Documentation

```
ixEthAccInit ( void )
```

Initialize the Ethernet Access Service.

Initialize the IXP425 Ethernet Access Service.

- Reentrant – no
- ISR Callable – no

This should be called once per module initialization.

**Precondition:**

The NPE must first be downloaded with the required microcode which supports all required features.

**Returns:**

IxEthAccStatus

◊ IX\_ETH\_ACC\_SUCCESS

◇ IX\_ETH\_ACC\_FAIL : Service has failed to initialize.

---

**ixEthAccMacInit ( IxEthAccPortId portId )**

Initialize the ethernet MAC settings.

\* – Reentrant – no

- ISR Callable – no

**Parameters:**

*portId* IxEthAccPortId

**Returns:**

IxEthAccStatus

◇ IX\_ETH\_ACC\_SUCCESS

◇ IX\_ETH\_ACC\_INVALID\_PORT :  
portId is invalid.

---

**ixEthAccMibIIStatsClear ( IxEthAccPortId portId )**

Clear the statistics maintained for a port. Clear the statistics maintained for a port.

- Reentrant – yes
- ISR Callable – no

**Precondition:**

**Parameters:**

*portId* IxEthAccPortId

**Returns:**

IxEthAccStatus

◇ IX\_ETH\_ACC\_SUCCESS

◇ IX\_ETH\_ACC\_FAIL : Invalid arguments.

◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.

◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

---

**ixEthAccMibIIStatsGet ( IxEthAccPortId portId,  
IxEthEthObjStats \* retStats  
)**

Return the statistics maintained for a port. Return the statistics maintained for a port.



- Reentrant – yes
- ISR Callable – no

**Precondition:**

**Parameters:**

*portId* *IxEthAccPortId*  
*retStats* ***IxEthEthObjStats***

**Note:**

Please note the user is responsible for cache coherency of the retStat buffer. The data is actually populated via the NPE's. As such cache safe memory should be used in the retStats argument.

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_FAIL : Invalid arguments.  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

---

```
ixEthAccMibIIStatsGetClear ( IxEthAccPortId    portId,
                             IxEthEthObjStats * retStats
                             )
```

Return and clear the statistics maintained for a port. Return and clear the statistics maintained for a port.

- Reentrant – yes
- ISR Callable – yes

**Precondition:**

**Parameters:**

*portId* *IxEthAccPortId*  
*retStats* ***IxEthEthObjStats***

**Note:**

Please note the user is responsible for cache coherency of the retStats buffer. The data is actually populated via the NPE's. As such cache safe memory should be used in the retStats argument.

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_FAIL : invalid arguments.  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

---

```

ixEthAccMiiReadRtn ( UINT8    phyAddr,
                     UINT8    phyReg,
                     UINT16 * value
                     )

```

Read a 16 bit value from a PHY.

Read a 16-bit word from a register of a MII-compliant PHY. Reading is performed through the MII management interface. This function returns when the read has successfully completed, or when a timeout has elapsed.

- Reentrant – no
- ISR Callable – no

**Precondition:**

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

**Parameters:**

*phyAddr*: the address of the Ethernet PHY (0–31)  
*phyReg*: the number of the MII register to read (0–31)  
*value*: the value read from the register

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_FAIL : failed to read register.

---

```

ixEthAccMiiStatsShow ( UINT32 phyAddr )

```

Display detailed information on a specified PHY.

Display the current values of the first eighth MII registers for a PHY,

- Reentrant – no
- ISR Callable – no

**Precondition:**

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

**Parameters:**

*phyAddr*: the address of the Ethernet PHY (0–31)

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_FAIL : invalid arguments.

---

```

IxEthAccMiiWriteRtn ( UINT8  phyAddr,
                      UINT8  phyReg,
                      UINT16 value
                      )

```

Write a 16 bit value to a PHY.

Write a 16-bit word from a register of a MII-compliant PHY. Writing is performed through the MII management interface. This function returns when the write has successfully completed, or when a timeout has elapsed.

- Reentrant – no
- ISR Callable – no

**Precondition:**

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

**Parameters:**

*phyAddr*: the address of the Ethernet PHY (0–31)  
*phyReg*: the number of the MII register to write (0–31)  
*value*: the value to write to the register

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_FAIL : failed to write register.

---

```

IxEthAccPortDisable ( IxEthAccPortId portId )

```

Disable a port.

This disables an Ethernet port for both Tx and Rx. Free MBufs are returned to the user via the registered callback when the port is disabled

- Reentrant – yes
- ISR Callable – yes

**Precondition:**

The port must be enabled with *ixEthAccPortEnable*, otherwise this function has no effect

**Parameters:**

*portId* : *IxEthAccPortId* : Port id to act upon.

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is not initialized

◇ IX\_ETH\_ACC\_MAC\_UNINITIALIZED : port MAC address is not initialized

---

```
ixEthAccPortDuplexModeGet ( IxEthAccPortId      portId,  
                           IxEthAccDuplexMode * mode  
                           )
```

Get the duplex mode for the MAC.

return the duplex configuration of the IXP425 MAC.

**Note:**

The configuration should match that provisioned on the PHY. See *ixEthAccDuplexModeSet*

- Reentrant – no
- ISR Callable – no

**Parameters:**

*portId* : *IxEthAccPortId*  
*\*mode* : *IxEthAccDuplexMode*

**Returns:**

IxEthAccStatus  
◇ IX\_ETH\_ACC\_SUCCESS  
◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

---

```
ixEthAccPortDuplexModeSet ( IxEthAccPortId      portId,  
                           IxEthAccDuplexMode mode  
                           )
```

Set the duplex mode for the MAC.

Configure the IXP425 MAC to either full or half duplex.

**Note:**

The configuration should match that provisioned on the PHY.

- Reentrant – no
- ISR Callable – no

**Parameters:**

*portId* : *IxEthAccPortId*  
*mode* : *IxEthAccDuplexMode*

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

---

**ixEthAccPortEnable ( IxEthAccPortId portId )**

Enable a port.

This enables an Ethernet port for both Tx and Rx.

- Reentrant – yes
- ISR Callable – yes

**Precondition:**

The port must first be initialized via *ixEthAccPortInit* and the MAC address must be set using *ixEthAccUnicastMacAddressSet* before enabling it. The rx and Tx Done callbacks registration via *ixEthAccPortTxDoneCallbackRegister* and *ixEthAccPortRxCallbackRegister* has to be done before enabling the traffic.

**Parameters:**

*portId* : IxEthAccPortId : Port id to act upon.

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is not initialized  
 ◇ IX\_ETH\_ACC\_MAC\_UNINITIALIZED : port MAC address is not initialized

---

**ixEthAccPortEnabledQuery ( IxEthAccPortId portId, BOOL \* enabled )**

Get the enabled state of a port.

Return the enabled state of the port.

- Reentrant – yes
- ISR Callable – yes

**Precondition:**

The port must first be initialized via  
*ixEthAccPortInit*

**Parameters:**

*portId* : *IxEthAccPortId* : Port id to act upon.  
*enabled* : BOOL : location to store the state of  
the port

**Returns:**

*IxEthAccStatus*  
◇ IX\_ETH\_ACC\_SUCCESS  
◇ IX\_ETH\_ACC\_INVALID\_PORT :  
portId is invalid

---

**ixEthAccPortInit ( *IxEthAccPortId* *portId* )**

Initialize an Ethernet MAC Port.

Initialize the NPE/Ethernet MAC hardware. Verify NPE downloaded and operational. The NPE shall be available for usage once this API returns. Verify that the Ethernet port is present before initializing

- Reentrant – no
- ISR Callable – no

This should be called once per mac device. The NPE/MAC shall be in disabled state after init.

**Precondition:**

The component must be initialized via *ixEthAccInit* The NPE must first be downloaded with the required microcode which supports all required features.

Dependant on Services: (Must be initialized before using this service may be initialized) ixNPEmh – NPE Message handling service. ixQmgr – Queue Manager component.

**Parameters:**

*portId* : *IxEthAccPortId*

**Returns:**

*IxEthAccStatus*  
◇ IX\_ETH\_ACC\_SUCCESS: if the ethernet port is not present, a warning will be issued.  
◇ IX\_ETH\_ACC\_FAIL : The NPE processor has failed to initialize.  
◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.

---

**ixEthAccPortMulticastAddressJoin ( *IxEthAccPortId* *portId*,  
*IxEthAccMacAddr* \* *macAddr*  
)**

ADD a multicast address to the MAC address table.

ADD a multicast address to the MAC address table.

**Note:**

Due to the operation of the Ethernet MAC multicast filtering mechanism, frames which do not have a multicast destination address which were provisioned via this API may be forwarded to the NPE's. This is a result of the hardware comparison algorithm used in the destination mac address logic within the Ethernet MAC.

See Also: IXP425 hardware development manual.

Other functions modify the MAC filtering

- ***ixEthAccPortMulticastAddressJoinAll()*** – all multicast frames are forwarded to the application
- ***ixEthAccPortMulticastAddressLeaveAll()*** – rollback the effects of ***ixEthAccPortMulticastAddressJoinAll()***
- ***ixEthAccPortMulticastAddressLeave()*** – unprovision a new filtering address
- ***ixEthAccPortMulticastAddressJoin()*** – provision a new filtering address
- ***ixEthAccPortPromiscuousModeSet()*** – all frames are forwarded to the application regardless of the multicast address provisioned
- ***ixEthAccPortPromiscuousModeClear()*** – frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant – no
- ISR Callable – no

**Parameters:**

*portId* – Ethernet port id.  
*\*macAddr* – Ethernet Mac address.

**Returns:**

**ixEthAccStatus**  
◊ **IX\_ETH\_ACC\_SUCCESS**  
◊ **IX\_ETH\_ACC\_FAIL** : Error writing to the MAC registers  
◊ **IX\_ETH\_ACC\_INVALID\_PORT** : *portId* is invalid.  
◊ **IX\_ETH\_ACC\_PORT\_UNINITIALIZED** : *portId* is un-initialized

---

**ixEthAccPortMulticastAddressJoinAll ( **ixEthAccPortId** *portId* )**

Filter all frames with multicast dest.

This function clears the MAC address table, and then sets the MAC to forward ALL multicast frames to the NPE. Specifically, it forwards all frames whose destination address has the LSB of the highest byte set (01:00:00:00:00:00). This bit is commonly referred to as the "multicast bit". Broadcast frames will still be forwarded.

Other functions modify the MAC filtering

- ***ixEthAccPortMulticastAddressJoinAll()*** – all multicast frames are forwarded to the application
- ***ixEthAccPortMulticastAddressLeaveAll()*** – rollback the effects of ***ixEthAccPortMulticastAddressJoinAll()***
- ***ixEthAccPortMulticastAddressLeave()*** – unprovision a new filtering address
- ***ixEthAccPortMulticastAddressJoin()*** – provision a new filtering address
- ***ixEthAccPortPromiscuousModeSet()*** – all frames are forwarded to the application regardless of the multicast address provisioned
- ***ixEthAccPortPromiscuousModeClear()*** – frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant – no
- ISR Callable – no

**Parameters:**

*portId* – Ethernet port id.

**Returns:**

**IxEthAccStatus**

◊ **IX\_ETH\_ACC\_SUCCESS**

◊ **IX\_ETH\_ACC\_INVALID\_PORT** : *portId* is invalid.

◊ **IX\_ETH\_ACC\_PORT\_UNINITIALIZED** : *portId* is un-initialized

---

```
ixEthAccPortMulticastAddressLeave ( IxEthAccPortId    portId,  
                                   IxEthAccMacAddr * macAddr  
                                   )
```

Remove a multicast address from the MAC address table.

Remove a multicast address from the MAC address table.

Other functions modify the MAC filtering

- ***ixEthAccPortMulticastAddressJoinAll()*** – all multicast frames are forwarded to the application
- ***ixEthAccPortMulticastAddressLeaveAll()*** – rollback the effects of ***ixEthAccPortMulticastAddressJoinAll()***
- ***ixEthAccPortMulticastAddressLeave()*** – unprovision a new filtering address
- ***ixEthAccPortMulticastAddressJoin()*** – provision a new filtering address
- ***ixEthAccPortPromiscuousModeSet()*** – all frames are forwarded to the application regardless of the multicast address provisioned
- ***ixEthAccPortPromiscuousModeClear()*** – frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.



- Reentrant – no
- ISR Callable – no

**Parameters:**

*portId* – Ethernet port id.  
*\*macAddr* – Ethernet Mac address.

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_NO\_SUCH\_ADDR : Failed if MAC address was not in the table.  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

---

**ixEthAccPortMulticastAddressLeaveAll ( IxEthAccPortId portId )**

Clear the MAC address table.

This function clears the MAC address table, and then sets the MAC as configured by the promiscuous mode current settings.

Other functions modify the MAC filtering

- **ixEthAccPortMulticastAddressJoinAll()** – all multicast frames are forwarded to the application
- **ixEthAccPortMulticastAddressLeaveAll()** – rollback the effects of **ixEthAccPortMulticastAddressJoinAll()**
- **ixEthAccPortMulticastAddressLeave()** – unprovision a new filtering address
- **ixEthAccPortMulticastAddressJoin()** – provision a new filtering address
- **ixEthAccPortPromiscuousModeSet()** – all frames are forwarded to the application regardless of the multicast address provisioned
- **ixEthAccPortPromiscuousModeClear()** – frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant – no
- ISR Callable – no

**Parameters:**

*portId* – Ethernet port id.

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

---

**ixEthAccPortMulticastAddressShow ( *IxEthAccPortId portId* )**

Display multicast address which have been configured using *ixEthAccMulticastAddressJoin* Display multicast address which have been configured using *ixEthAccMulticastAddressJoin*.

\* – Reentrant – yes

- ISR Callable – no

**Parameters:**

*portId* – Ethernet port id.

**Returns:**

void

---

**ixEthAccPortPromiscuousModeClear ( *IxEthAccPortId portId* )**

Put the Ethernet MAC device in non-promiscuous mode.

In non-promiscuous mode the MAC will filter all frames other than destination MAC address which matches the following criteria:

- Unicast address provisioned via *ixEthAccUnicastMacAddressSet*
- All broadcast frames.
- Multicast addresses provisioned via *ixEthAccMulticastAddressJoin*

See also: *ixEthAccPortPromiscuousModeSet*

Other functions modify the MAC filtering

- *ixEthAccPortMulticastAddressJoinAll()* – all multicast frames are forwarded to the application
- *ixEthAccPortMulticastAddressLeaveAll()* – rollback the effects of *ixEthAccPortMulticastAddressJoinAll()*
- *ixEthAccPortMulticastAddressLeave()* – unprovision a new filtering address
- *ixEthAccPortMulticastAddressJoin()* – provision a new filtering address
- *ixEthAccPortPromiscuousModeSet()* – all frames are forwarded to the application regardless of the multicast address provisioned
- *ixEthAccPortPromiscuousModeClear()* – frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* – Ethernet port id.

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

---

**ixEthAccPortPromiscuousModeSet ( IxEthAccPortId portId )**

Put the MAC device in promiscuous mode.

If the device is in promiscuous mode then all received frames shall be forwarded to the NPE for processing.

See also: *ixEthAccPortPromiscuousModeClear*

Other functions modify the MAC filtering

- *ixEthAccPortMulticastAddressJoinAll()* – all multicast frames are forwarded to the application
- *ixEthAccPortMulticastAddressLeaveAll()* – rollback the effects of *ixEthAccPortMulticastAddressJoinAll()*
- *ixEthAccPortMulticastAddressLeave()* – unprovision a new filtering address
- *ixEthAccPortMulticastAddressJoin()* – provision a new filtering address
- *ixEthAccPortPromiscuousModeSet()* – all frames are forwarded to the application regardless of the multicast address provisioned
- *ixEthAccPortPromiscuousModeClear()* – frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* – Ethernet port id.

**Returns:**

IxEthAccStatus  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

---

**ixEthAccPortRxCallbackRegister ( IxEthAccPortId portId,**  
**IxEthAccPortRxCallback rxCallbackFn,**  
**UINT32 callbackTag**  
**)**

The function registered through this function shall be called once per received Ethernet frame.

This function will dispatch a predefined number of frames to the user level via the provided function. The invocation shall be made for each frame dequeued from the Ethernet QM queue. The user is required to free any MBUF's supplied via this callback. In addition the registered callback must free up MBUF's from the receive free queue when the port is disabled

If called several times the latest callback shall be registered for a particular port.

- Reentrant – yes
- ISR Callable – yes

**Parameters:**

*portId* – Register callback for a particular MAC device.  
*rxCallbackFn* – *IxEthAccRxCallbackFn* – Function to be called when Ethernet frames are available.  
*callbackTag* – This tag shall be provided to the callback function.

**Returns:**

*IxEthAccStatus*  
◊ *IX\_ETH\_ACC\_SUCCESS*  
◊ *IX\_ETH\_ACC\_INVALID\_PORT* : *portId* is invalid.  
◊ *IX\_ETH\_ACC\_PORT\_UNINITIALIZED* : *portId* is un-initialized  
◊ *IX\_ETH\_ACC\_INVALID\_ARG* : An argument other than *portId* is invalid.

---

**ixEthAccPortRxFrameAppendFCSDisable ( *IxEthAccPortId portId* )**

Disable the appending of Ethernet FCS to all frames submitted to this port.

Do not forward the FCS portion of the received Ethernet frame to the user. The FCS is stripped from the receive buffer. Frame FCS validity checks are still carried out on all received frames. This is the default behavior of the component. Do not change this behaviour while the port is enabled.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* : *IxEthAccPortId*

**Returns:**

*IxEthAccStatus*  
◊ *IX\_ETH\_ACC\_SUCCESS*  
◊ *IX\_ETH\_ACC\_INVALID\_PORT* : *portId* is invalid.  
◊ *IX\_ETH\_ACC\_PORT\_UNINITIALIZED* : *portId* is un-initialized

---

**ixEthAccPortRxFrameAppendFCSEnable ( *IxEthAccPortId portId* )**

Forward frames with FCS included in the receive buffer to the user.

Enable the appending of Ethernet FCS to all frames submitted to this port. This is the default behavior of the access component. The Frame length recieved will include the FCS. ie. A minimum sized ethernet frame shall have a length of 64bytes.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* : *IxEthAccPortId*

**Returns:**

*IxEthAccStatus*

- ◇ IX\_ETH\_ACC\_SUCCESS
  - ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.
  - ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized
- 

```
ixEthAccPortRxFreeReplenish ( IxEthAccPortId portId,  
                             IX_MBUF *    buffer  
                             )
```

This function provides buffers for the Ethernet receive path.

This component does not have a buffer management mechanisms built in. All Rx buffers must be supplied to it via this interface.

- Reentrant – yes
- ISR Callable – yes

**Parameters:**

*portId* – Provide buffers only to specific Rx MAC.

*buffer* – Provide an MBUF to the Ethernet receive mechanism. Buffers size smaller than IX\_ETHACC\_RX\_MBUF\_MIN\_SIZE may result in poor performances and excessive buffer chaining. Buffers larger than this size may be suitable for jumbo frames. Chained packets are not supported and the field IX\_MBUF\_NEXT\_PKT\_IN\_CHAIN\_PTR must be NULL.

**Returns:**

*IxEthAccStatus*

- ◇ IX\_ETH\_ACC\_SUCCESS
- ◇ IX\_ETH\_ACC\_FAIL : Buffer has was not able to queue the buffer in the receive service.
- ◇ IX\_ETH\_ACC\_FAIL : Buffer size is less than IX\_ETHACC\_RX\_MBUF\_MIN\_SIZE
- ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.
- ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

**Note:**

If the buffer replenish operation fails it is the responsibility of the user to free the buffer.

Sufficient buffers must be supplied to the component to maintain receive throughput and avoid rx buffer underflow conditions. To meet this goal, It is expected that the user preload the component with a sufficient number of buffers prior to enabling the NPE Ethernet receive path. The recommended minimum number of buffers is 8.

For maximum performances, the mbuf size should be greater than the maximum frame size (Ethernet header, payload and FCS) + 64. Supplying smaller mbufs to the service results in mbuf chaining and degraded performances. The recommended size is *IX\_ETHACC\_RX\_MBUF\_MIN\_SIZE*, which is enough to take care of 802.3 frames and "baby jumbo" frames without chaining, and "jumbo" frame within chaining.

Buffers may not be filled up to their length. The firmware will fill them up to the previous 64 bytes boundary. The user has to be aware that the length of the received mbufs may be smaller than the length of the supplied mbufs.

**Warning:**

This function will only check the parameters if the NDEBUG flag is not defined. Turning on the argument checking (disabled by default) will result in a lower EthAcc performance as this function is part of the data path.

---

```

ixEthAccPortTxDoneCallbackRegister ( IxEthAccPortId          portId,
                                     IxEthAccPortTxDoneCallback txCallbackFn,
                                     UINT32                     callbackTag
                                     )

```

This function registers a callback function to facilitate the return of transmit buffers to the user.

This function registers the transmit buffer done function callback for a particular port.

The registered callback function is called once the previously submitted buffer is no longer required by this service. It may be returned upon successful transmission of the frame or shutdown of port prior to submission. The calling of this registered function is not a guarantee of successful transmission of the buffer.

If called several times the latest callback shall be registered for a particular port.

- Reentrant – yes
- ISR Callable – yes

**Precondition:**

The port must be initialized via *ixEthAccPortInit*

**Parameters:**

- portId* – Register callback for a particular MAC device.
- txCallbackFn* – *IxEthAccTxBufferDoneCallbackFn* – Function to be called to return transmit buffers to the user.
- callbackTag* – This tag shall be provided to the callback function.

**Returns:**

IxEthAccStatus

- ◊ IX\_ETH\_ACC\_SUCCESS
  - ◊ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.
  - ◊ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized
  - ◊ IX\_ETH\_ACC\_INVALID\_ARG : An argument other than portId is invalid.
- 

**ixEthAccPortTxFrameAppendFCSDisable ( IxEthAccPortId portId )**

Disable the appending of Ethernet FCS to all frames submitted to this port.

Disable the appending of Ethernet FCS to all frames submitted to this port. This is not the default behavior of the access component. Note: Since the FCS is not appended to the frame it is expected that the frame submitted to the component includes a valid FCS at the end of the data, although this will not be validated. The component shall forward the frame to the Ethernet MAC WITHOUT modification. Do not change this behaviour while the port is enabled.

**Note:**

Tx FCS append is not disabled while Tx padding is enabled.

**See also:**

**ixEthAccPortTxFrameAppendPaddingEnable**

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* : IxEthAccPortId

**Returns:**

IxEthAccStatus

- ◊ IX\_ETH\_ACC\_SUCCESS
  - ◊ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.
  - ◊ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized
- 

**ixEthAccPortTxFrameAppendFCSEnable ( IxEthAccPortId portId )**

Enable the appending of Ethernet FCS to all frames submitted to this port.

Enable the appending of Ethernet FCS to all frames submitted to this port. This is the default behavior of the access component. Do not change this behaviour while the port is enabled.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* : *IxEthAccPortId*

**Returns:**

*IxEthAccStatus*  
◊ *IX\_ETH\_ACC\_SUCCESS*  
◊ *IX\_ETH\_ACC\_INVALID\_PORT* : *portId* is invalid.  
◊ *IX\_ETH\_ACC\_PORT\_UNINITIALIZED* : *portId* is un-initialized

---

*ixEthAccPortTxFrameAppendPaddingDisable* ( ***IxEthAccPortId*** *portId* )

Disable the appending of padding bytes to the runt frames submitted to this port.

Disable the appending of padding bytes to runt frames submitted to this port. This is not the default behavior of the access component.

**Warning:**

Do not change this behaviour while the port is enabled.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* : *IxEthAccPortId*

**Returns:**

*IxEthAccStatus*  
◊ *IX\_ETH\_ACC\_SUCCESS*  
◊ *IX\_ETH\_ACC\_INVALID\_PORT* : *portId* is invalid.  
◊ *IX\_ETH\_ACC\_PORT\_UNINITIALIZED* : *portId* is un-initialized

---

*ixEthAccPortTxFrameAppendPaddingEnable* ( ***IxEthAccPortId*** *portId* )

Enable the appending of padding bytes to runt frames submitted to this port.

Enable the appending of up to 60 null-bytes to runt frames submitted to this port. This is the default behavior of the access component.

**Warning:**

Do not change this behaviour while the port is enabled.

**Note:**

When Tx padding is enabled, Tx FCS generation is turned on

**See also:**

*ixEthAccPortTxFrameAppendFCSDisable*



- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* : *IxEthAccPortId*

**Returns:**

*IxEthAccStatus*

◇ IX\_ETH\_ACC\_SUCCESS

◇ IX\_ETH\_ACC\_INVALID\_PORT : *portId* is invalid.

◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : *portId* is un-initialized

```
ixEthAccPortTxFrameSubmit ( IxEthAccPortId    portId,
                             IX_MBUF *          buffer,
                             IxEthAccTxPriority priority
                             )
```

This function shall be used to submit Mbufs buffers for transmission on a particular MAC device.

This function shall be used to submit Mbufs buffers for transmission on a particular MAC device.

When the frame is transmitted, the buffer shall be returned thru the callback *IxEthAccPortTxDoneCallback*.

The only alterations made to the buffer are associated with the next packet chain pointer. This is used to internally queue frames in the service when submitting faster than the network line rate. In case of over=Submitting, the order of the frames on the network may be modified.

Buffers shall be not queued for transmission if the port is disabled. The port can be enabled using *ixEthAccPortEnable*

- Reentrant – yes
- ISR Callable – yes

**Precondition:**

*ixEthAccPortTxDoneCallbackRegister* must be called to register a function to allow this service to return the buffer to the calling service.

**Note:**

If the buffer submit fails for any reason the user has retained ownership of the buffer.

**Parameters:**

*portId* – MAC port ID to transmit Ethernet frame on.

*buffer* – pointer to an Mbuf formatted buffer. Chained buffers are supported for transmission.

Chained packets are not supported and the field

IX\_MBUF\_NEXT\_PKT\_IN\_CHAIN\_PTR must be NULL.

*priority* – *IxEthAccTxPriority*

**Returns:**

**IxEthAccStatus**

- ◊ **IX\_ETH\_ACC\_SUCCESS**
  - ◊ **IX\_ETH\_ACC\_FAIL** : Failed to queue frame for transmission.
  - ◊ **IX\_ETH\_ACC\_INVALID\_PORT** : portId is invalid.
  - ◊ **IX\_ETH\_ACC\_PORT\_UNINITIALIZED** : portId is un-initialized
- 

**ixEthAccPortUnicastAddressShow ( **IxEthAccPortId** portId )**

Display unicast address has been configured using *ixEthAccUnicastMacAddressSet*.

Display unicast address has been configured using *ixEthAccUnicastMacAddressSet*. Display also the MAC filter used

Other functions modify the MAC filtering

- ***ixEthAccPortMulticastAddressJoinAll()*** – all multicast frames are forwarded to the application
- ***ixEthAccPortMulticastAddressLeaveAll()*** – rollback the effects of *ixEthAccPortMulticastAddressJoinAll()*
- ***ixEthAccPortMulticastAddressLeave()*** – unprovision a new filtering address
- ***ixEthAccPortMulticastAddressJoin()*** – provision a new filtering address
- ***ixEthAccPortPromiscuousModeSet()*** – all frames are forwarded to the application regardless of the multicast address provisioned
- ***ixEthAccPortPromiscuousModeClear()*** – frames are forwarded to the application following the multicast address provisioned

In all cases, unicast and broadcast addresses are forwarded to the application.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* – Ethernet port id.

**Returns:**

void

---

**ixEthAccPortUnicastMacAddressGet ( **IxEthAccPortId** portId,  
  **IxEthAccMacAddr** \* macAddr  
  )**

Get unicast MAC address for a particular MAC port.

**Precondition:**

The MAC address must first be set via *ixEthAccMacPromiscuousModeSet* If the MAC address has not been set, the function will return a **IX\_ETH\_ACC\_MAC\_UNINITIALIZED** status

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* – Ethernet port id.  
*\*macAddr* – Ethernet MAC address.

**Returns:**

**IxEthAccStatus**  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_MAC\_UNINITIALIZED : port MAC address is not initialized.  
 ◇ IX\_ETH\_ACC\_FAIL : macAddr is invalid.

---

```
ixEthAccPortUnicastMacAddressSet ( IxEthAccPortId    portId,
                                   IxEthAccMacAddr * macAddr
                                   )
```

Configure unicast MAC address for a particular port.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portId* – Ethernet port id.  
*\*macAddr* – Ethernet Mac address.

**Returns:**

**IxEthAccStatus**  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_INVALID\_PORT : portId is invalid.  
 ◇ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : portId is un-initialized

---

```
ixEthAccStatsShow ( IxEthAccPortId portId )
```

Display a ports statistics on the standard io console using printf. Display a ports statistics on the standard io console using printf.

- Reentrant – no
- ISR Callable – no

**Precondition:**

**Parameters:**

*portId* *IxEthAccPortId*

**Returns:**

void

---

```
ixEthAccTxSchedulingDisciplineSet ( IxEthAccPortId          portId,  
                                   IxEthAccTxSchedulerDiscipline sched  
                                   )
```

Set the port scheduling to one of *IxEthAccTxSchedulerDiscipline*. Set the port scheduling to one of *IxEthAccTxSchedulerDiscipline*.

The default behavior of the component is *FIFO\_NO\_PRIORITY*.

- Reentrant – yes
- ISR Callable – yes

**Precondition:**

**Parameters:**

*portId* : *IxEthAccPortId*

*sched* : *IxEthAccTxSchedulerDiscipline*

**Returns:**

*IxEthAccStatus*

- ◊ IX\_ETH\_ACC\_SUCCESS : Set appropriate discipline.
  - ◊ IX\_ETH\_ACC\_FAIL : Invalid/unsupported discipline.
  - ◊ IX\_ETH\_ACC\_INVALID\_PORT : *portId* is invalid.
  - ◊ IX\_ETH\_ACC\_PORT\_UNINITIALIZED : *portId* is un-initialized
- 

```
ixEthAccUnload ( void )
```

Unload the Ethernet Access Service.

\* – Reentrant – no

- ISR Callable – no

**Returns:**

void

---

# IXP425 Ethernet Database (IxEthDB) API

ethDB is a library that does provides a MAC address database learning/filtering capability

## Data Structures

```
struct IxEthDBMacAddr  
The IEEE 802.3 Ethernet MAC address structure.
```

## Defines

```
#define INLINE  
#define IX_ETH_DB_PRIVATE  
#define IX_ETH_DB_PUBLIC  
#define IX_ETH_DB_PORT_ID_TO_NPE(id)  
port ID => message handler NPE id conversion (0 => NPE_B, 1 => NPE_C)  
  
#define IX_ETH_DB_NPE_TO_PORT_ID(npe)  
message handler NPE id => port ID conversion (NPE_B => 0, NPE_C => 1)  
  
#define IX_IEEE803_MAC_ADDRESS_SIZE  
The size of the MAC address.  
  
#define IX_ETH_DB_MAINTENANCE_TIME  
The ixEthDBDatabaseMaintenance must be called by the user at a frequency of  
IX_ETH_DB_MAINTENANCE_TIME.  
  
#define IX_ETH_DB_LEARNING_ENTRY_AGE_TIME  
The define specifies the filtering database age entry time. Static entries older than  
IX_ETH_DB_LEARNING_ENTRY_AGE_TIME +/-  
IX_ETH_DB_MAINTENANCE_TIME shall be removed.
```

## Typedefs

```
typedef UINT32 IxEthDBPortId  
Definition of an IXP425 port.  
  
typedef UINT32 IxEthDBPortMap  
Port dependency map definition.
```

## Enumerations

```
enum IxEthDBStatus {  
    IX_ETH_DB_SUCCESS,  
    IX_ETH_DB_FAIL,  
    IX_ETH_DB_INVALID_PORT,  
    IX_ETH_DB_PORT_UNINITIALIZED,  
    IX_ETH_DB_MAC_UNINITIALIZED,  
    IX_ETH_DB_INVALID_ARG,  
    IX_ETH_DB_NO_SUCH_ADDR,  
    IX_ETH_DB_NOMEM,  
    IX_ETH_DB_BUSY,  
    IX_ETH_DB_END  
}  
Ethernet database status.
```

## Functions

**IX\_ETH\_DB\_PUBLIC**

**IxEthDBStatus** **ixEthDBInit** (void)

*Initializes the Ethernet learning/filtering database.*

**IX\_ETH\_DB\_PUBLIC**

void **ixEthDBPortInit** (**IxEthDBPortId** portID)

*Initializes a port.*

**IX\_ETH\_DB\_PUBLIC**

**IxEthDBStatus** **ixEthDBPortEnable** (**IxEthDBPortId** portID)

*enable a port*

**IX\_ETH\_DB\_PUBLIC**

**IxEthDBStatus** **ixEthDBPortDisable** (**IxEthDBPortId** portID)

*disable processing on a port*

**IX\_ETH\_DB\_PUBLIC** **ixEthDBPortAddressSet** (**IxEthDBPortId** portID, **IxEthDBMacAddr**

**IxEthDBStatus** \*macAddr)

*set the port MAC address*

**IxEthDBStatus** **ixEthDBFilteringPortMaximumFrameSizeSet** (**IxEthDBPortId** portID,

UINT32 maximumFrameSize)

*Set the maximum frame size supported on the given port ID.*

**IxEthDBStatus** **ixEthDBFilteringStaticEntryProvision** (**IxEthDBPortId** portID,

**IxEthDBMacAddr** \*macAddr)

*Populate the Ethernet learning/filtering database with a static MAC address.*

**IxEthDBStatus** **ixEthDBFilteringDynamicEntryProvision** (**IxEthDBPortId** portID,

**IxEthDBMacAddr** \*macAddr)

*Populate the Ethernet learning/filtering database with a dynamic MAC address.*

**IxEthDBStatus ixEthDBFilteringEntryDelete (IxEthDBMacAddr \*macAddr)**

*Remove a MAC address entry from the Ethernet learning/filtering database.*

**IxEthDBStatus ixEthDBFilteringPortSearch (IxEthDBPortId portID, IxEthDBMacAddr \*macAddr)**

*Search the Ethernet learning/filtering database for the given MAC address and port ID.*

**IxEthDBStatus ixEthDBFilteringDatabaseSearch (IxEthDBPortId \*portID, IxEthDBMacAddr \*macAddr)**

*Search the Ethernet learning/filtering database for a MAC address and return the port ID.*

**IxEthDBStatus ixEthDBFilteringPortUpdatingSearch (IxEthDBPortId \*portID, IxEthDBMacAddr \*macAddr)**

*Search the filtering database for a MAC address, return the port ID and reset the record age.*

**IxEthDBStatus ixEthDBPortAgingDisable (IxEthDBPortId port)**

*Disable the aging function for a specific port.*

**IxEthDBStatus ixEthDBPortAgingEnable (IxEthDBPortId portID)**

*Enable the aging function for a specific port.*

**void ixEthDBDatabaseMaintenance (void)**

*Performs a maintenance operation on the Ethernet learning/filtering database.*

**IxEthDBStatus ixEthDBFilteringDatabaseShow (IxEthDBPortId portID)**

*This function displays the Mac Ethernet MAC address filtering tables.*

**void ixEthDBFilteringDatabaseShowAll (void)**

*Displays the MAC address recorded in the filtering database for all registered ports (see **IxEthDBPortDefs.h**), grouped by port ID.*

---

## Detailed Description

ethDB is a library that does provides a MAC address database learning/filtering capability

---

## Define Documentation

```
#define IX_ETH_DB_LEARNING_ENTRY_AGE_TIME
```

The define specifies the filtering database age entry time. Static entries older than IX\_ETH\_DB\_LEARNING\_ENTRY\_AGE\_TIME +/- IX\_ETH\_DB\_MAINTENANCE\_TIME shall be removed.

Definition at line **421** of file **IxEthDB.h**.

```
#define IX_ETH_DB_MAINTENANCE_TIME
```

The **ixEthDBDatabaseMaintenance** must be called by the user at a frequency of IX\_ETH\_DB\_MAINTENANCE\_TIME.

Definition at line **410** of file **IxEthDB.h**.

```
#define IX_ETH_DB_NPE_TO_PORT_ID ( npe )
```

message handler NPE id => port ID conversion (NPE\_B => 0, NPE\_C => 1)

Definition at line **76** of file **IxEthDB.h**.

```
#define IX_ETH_DB_PORT_ID_TO_NPE ( id )
```

port ID => message handler NPE id conversion (0 => NPE\_B, 1 => NPE\_C)

Definition at line **70** of file **IxEthDB.h**.

```
#define IX_IEEE803_MAC_ADDRESS_SIZE
```

The size of the MAC address.

Definition at line **82** of file **IxEthDB.h**.

---

## Typedef Documentation

```
typedef UINT32 IxEthDBPortId
```

Definition of an IXP425 port.

Definition at line **120** of file **IxEthDB.h**.

```
typedef UINT32 IxEthDBPortMap
```

Port dependency map definition.

Definition at line **127** of file **IxEthDB.h**.

---



# Enumeration Type Documentation

enum IxEthDBStatus

Ethernet database status.

**Enumeration values:**

<i>IX_ETH_DB_SUCCESS</i>	Return success.
<i>IX_ETH_DB_FAIL</i>	Return fail.
<i>IX_ETH_DB_INVALID_PORT</i>	invalid port
<i>IX_ETH_DB_PORT_UNINITIALIZED</i>	port uninitialized
<i>IX_ETH_DB_MAC_UNINITIALIZED</i>	MAC uninitialized.
<i>IX_ETH_DB_INVALID_ARG</i>	invalid arg
<i>IX_ETH_DB_NO_SUCH_ADDR</i>	Address not found for search or delete operations.
<i>IX_ETH_DB_NOMEM</i>	Learning database memory full.
<i>IX_ETH_DB_BUSY</i>	Learning database is busy.
<i>IX_ETH_DB_END</i>	Database browser passed the end.

Definition at line 88 of file **IxEthDB.h**.

---

## Function Documentation

void ixEthDBDatabaseMaintenance ( void )

Performs a maintenance operation on the Ethernet learning/filtering database.

In order to perform a database maintenance this function must be called every seconds. It should be called regardless of whether learning is enabled or not.

- Reentrant – no
- ISR Callable – no

**Return values:**

*void*

```
IxEthDBStatus ixEthDBFilteringDatabaseSearch ( IxEthDBPortId * portID,  
                                                IxEthDBMacAddr * macAddr  
                                                )
```

Search the Ethernet learning/filtering database for a MAC address and return the port ID.

Searches the database for a MAC address. The function returns the portID for the MAC address record, if found. If no match is found the function returns IX\_ETH\_DB\_NO\_SUCH\_ADDR. The portID is only valid if the function finds a match.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portID* port ID the address belongs to (populated only on a successful search)  
*macAddr* MAC address to search for

**Return values:**

*IX\_ETH\_DB\_SUCCESS* the record exists in the database  
*IX\_ETH\_DB\_NO\_SUCH\_ADDR* the record was not found in the database

**IxEthDBStatus** ixEthDBFilteringDatabaseShow ( **IxEthDBPortId** *portID* )

This function displays the Mac Ethernet MAC address filtering tables.

It displays the MAC address, port ID, entry type (dynamic/static),and age for the given port ID.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portID* port ID to display the MAC address entries

**Return values:**

*IX\_ETH\_DB\_SUCCESS* operation completed successfully  
*IX\_ETH\_DB\_INVALID\_PORT* portID is invalid  
*IX\_ETH\_DB\_PORT\_UNINITIALIZED* port ID is not initialized

**void** ixEthDBFilteringDatabaseShowAll ( **void** )

Displays the MAC address recorded in the filtering database for all registered ports (see **IxEthDBPortDefs.h**), grouped by port ID.

**Return values:**

*void*      ◇ Reentrant  
              – yes  
              ◇ ISR  
              Callable –  
              no

**IxEthDBStatus** ixEthDBFilteringDynamicEntryProvision ( **IxEthDBPortId** *portID*,  
    **IxEthDBMacAddr** \* *macAddr*  
    )

Populate the Ethernet learning/filtering database with a dynamic MAC address.

Populates the Ethernet learning/filtering database with a dynamic MAC address. This entry will be subject to normal aging function, if aging is enabled on its port. If there is an entry (static or dynamic) with the same MAC address on any port this entry will take precedence. Any other entry with the same MAC address will be removed.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portID* port ID to add the dynamic address to  
*macAddr* static MAC address to add

**Return values:**

*IX\_ETH\_DB\_SUCCESS* the add was successful  
*IX\_ETH\_DB\_FAIL* failed to populate the database entry  
*IX\_ETH\_DB\_INVALID\_PORT* portID is invalid  
*IX\_ETH\_DB\_PORT\_UNINITIALIZED* port is not initialized

**IxEthDBStatus** ixEthDBFilteringEntryDelete ( **IxEthDBMacAddr** \* *macAddr* )

Remove a MAC address entry from the Ethernet learning/filtering database.

\* – Reentrant – yes

- ISR Callable – no

**Parameters:**

*macAddr* MAC address to remove

**Return values:**

*IX\_ETH\_DB\_SUCCESS* the removal was successful  
*IX\_ETH\_DB\_NO\_SUCH\_ADDR* failed to remove the address (not in the database)

**IxEthDBStatus** ixEthDBFilteringPortMaximumFrameSizeSet ( **IxEthDBPortId** *portID*,  
UINT32 *maximumFrameSize*  
)

Set the maximum frame size supported on the given port ID.

This functions set the maximum frame size supported on a specific port ID

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portID* port ID to configure  
*maximumFrameSize* maximum frame size to configure

**Return values:**

*IX\_ETH\_DB\_SUCCESS* the port is configured  
*IX\_ETH\_DB\_INVALID\_PORT* portID is invalid  
*IX\_ETH\_DB\_INVALID\_ARG* parameter is out of range

**Note:**

This maximum frame size is used to filter the frames based on their destination addresses and the capabilities of the destination port. The maximum value that can be set for a NPE port is 16320. (IX\_ETHNPE\_ACC\_FRAME\_LENGTH\_MAX)

```
IxEthDBStatus ixEthDBFilteringPortSearch ( IxEthDBPortId portID,
                                           IxEthDBMacAddr * macAddr
                                           )
```

Search the Ethernet learning/filtering database for the given MAC address and port ID.

This function searches the database for a specific port ID and MAC address. Both the port ID and the MAC address have to match in order for the record to be reported as found.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portID* port ID to search for  
*macAddr* MAC address to search for

**Return values:**

*IX\_ETH\_DB\_SUCCESS* the record exists in the database  
*IX\_ETH\_DB\_NO\_SUCH\_ADDR* the record was not found in the database  
*IX\_ETH\_DB\_INVALID\_PORT* portID is invalid  
*IX\_ETH\_DB\_PORT\_UNINITIALIZED* port ID is not initialized

```
IxEthDBStatus ixEthDBFilteringPortUpdatingSearch ( IxEthDBPortId * portID,
                                                    IxEthDBMacAddr * macAddr
                                                    )
```

Search the filtering database for a MAC address, return the port ID and reset the record age.

Searches the database for a MAC address. The function returns the portID for the MAC address record and resets the entry age to 0, if found. If no match is found the function returns IX\_ETH\_DB\_NO\_SUCH\_ADDR. The portID is only valid if the function finds a match.

- Reentrant – yes
- ISR Callable – no

***Return values:***

*IX\_ETH\_DB\_SUCCESS*            the MAC address was found

*IX\_ETH\_DB\_NO\_SUCH\_ADDR* the MAC address was not found

```
IxEthDBStatus ixEthDBFilteringStaticEntryProvision ( IxEthDBPortId      portID,
                                                         IxEthDBMacAddr * macAddr
                                                         )
```

Populate the Ethernet learning/filtering database with a static MAC address.

Populates the Ethernet learning/filtering database with a static MAC address. The entry will not be subject to aging. If there is an entry (static or dynamic) with the corresponding MAC address on any port this entry will take precedence. Any other entry with the same MAC address will be removed.

- Reentrant – yes
- ISR Callable – no

***Parameters:***

*portID* port ID to add the static address to

*macAddr* static MAC address to add

***Return values:***

*IX\_ETH\_DB\_SUCCESS* the add was successful

<i>IX_ETH_DB_FAIL</i>	failed to populate the database entry
-----------------------	---------------------------------------

*IX\_ETH\_DB\_INVALID\_PORT* portID is invalid

*IX\_ETH\_DB\_PORT\_UNINITIALIZED* port is not initialized

**IxEthDBStatus** ixEthDBInit ( void )

Initializes the Ethernet learning/filtering database.

***Return values:***

*IX\_ETH\_DB\_SUCCESS* initialization was successful

<i>IX_ETH_DB_FAIL</i>	initialization failed (OSSL error)
-----------------------	---------------------------------------

```

IxEthDBStatus ixEthDBPortAddressSet ( IxEthDBPortId      portID,
                                         IxEthDBMacAddr * macAddr
                                         )

```

set the port MAC address

This function is to be called from the Ethernet Access component top-level `ixEthDBUnicastAddressSet()`. Event processing cannot be enabled for a port until its MAC address has been set.

**Parameters:**

*portID* ID of the port whose MAC address is set  
*macAddr* port MAC address

**Return values:**

*IX\_ETH\_DB\_SUCCESS* MAC address was set successfully  
*IX\_ETH\_DB\_FAIL* MAC address was not set due to a message handler failure  
*IX\_ETH\_DB\_INVALID\_PORT* if the port is not an Ethernet NPE

**See also:**

**IxEthDBPortDefs.h** for port definitions

**IxEthDBStatus** ixEthDBPortAgingDisable ( **IxEthDBPortId** *port* )

Disable the aging function for a specific port.

\* – Reentrant – yes

- ISR Callable – no

**Parameters:**

*portID* port ID to disable aging on

**Return values:**

*IX\_ETH\_DB\_SUCCESS* aging disabled successfully  
*IX\_ETH\_DB\_INVALID\_PORT* portID is invalid  
*IX\_ETH\_DB\_PORT\_UNINITIALIZED* port ID is not initialized

**IxEthDBStatus** ixEthDBPortAgingEnable ( **IxEthDBPortId** *portID* )

Enable the aging function for a specific port.

Enables the aging of dynamic MAC address entries stored in the learning/filtering database

**Note:**

The aging function relies on the **ixEthDBDatabaseMaintenance** being called with a period of **IX\_ETH\_DB\_MAINTENANCE\_TIME** seconds.

- Reentrant – yes
- ISR Callable – no

**Parameters:**

*portID* port ID to enable aging on

**Return values:**

*IX\_ETH\_DB\_SUCCESS* aging enabled successfully

*IX\_ETH\_DB\_INVALID\_PORT* portID is invalid  
*IX\_ETH\_DB\_PORT\_UNINITIALIZED* port ID is not initialized

### **IxEthDBStatus** ixEthDBPortDisable ( **IxEthDBPortId** *portID* )

disable processing on a port

This function is called automatically from the Ethernet Access component top-level portDisable() routine and should be manually called for any user-defined port (any port that is not one of the two Ethernet NPEs).

Note: After Ethernet NPEs are disabled they are stopped therefore when re-enabled they need to be reset, downloaded with microcode and started. For learning to restart working the user needs to call again ixEthAccPortUnicastMacAddressSet or ixEthDBUnicastAddressSet with the respective port MAC address. Residual MAC addresses learnt before the port was disable will take up to IX\_ETH\_DB\_MAINTENANCE\_TIME to be downloaded in the NPE MAC tree after learning is restarted on the port. The dynamic addresses do not disappear after the disable-enable sequence. They also do not age during the time the port is disabled.

#### **Parameters:**

*portID* ID of the port to disable processing on

#### **Return values:**

*IX\_ETH\_DB\_SUCCESS* if disabling is successful  
*IX\_ETH\_DB\_FAIL* if the disabling was not successful due to a message handler error

### **IxEthDBStatus** ixEthDBPortEnable ( **IxEthDBPortId** *portID* )

enable a port

This function is called automatically from the Ethernet Access component top-level portEnable() routine and should be manually called for any user-defined port (any port that is not one of the two Ethernet NPEs).

#### **Parameters:**

*portID* ID of the port to enable processing on

#### **Return values:**

*IX\_ETH\_DB\_SUCCESS* if enabling is successful  
*IX\_ETH\_DB\_FAIL* if the enabling was not successful due to a message handler error  
*IX\_ETH\_DB\_MAC\_UNINITIALIZED* the MAC address of this port was not initialized (only for Ethernet NPEs)

#### **Precondition:**

**ixEthDBPortAddressSet()** needs to be called prior to enabling the port events for Ethernet NPEs

#### **See also:**

## **ixEthDBPortAddressSet()**

**IxEthDBPortDefs.h** for port definitions

```
void ixEthDBPortInit ( IxEthDBPortId portID )
```

Initializes a port.

This function is called automatically by the Ethernet Access component top-level portInit() routine and should be manually called for any user-defined port (any port that is not one of the two Ethernet NPEs).

### ***Parameters:***

*portID* ID of the port to be initialized

### ***Return values:***

*void*

### ***See also:***

**IxEthDBPortDefs.h** for port definitions



# IXP425 Ethernet Database Port Definitions (IxEthDBPortDefs)

IXP425 Ethernet Port Definitions for private MAC learning API.

## Data Structures

struct **IxEthDBPortDefinition**

*Port Definition – a structure contains the Port type and capabilities.*

## Defines

#define **IX\_ETH\_DB\_NUMBER\_OF\_PORTS**

*number of supported ports*

#define **IX\_ETH\_DB\_PORTS\_ASSERTION**

*catch invalid port definitions (<2) with a compile–time assertion resulting in a duplicate case error.*

#define **COMPLETE\_ETH\_PORT\_MAP**

*complete set of ports in use*

#define **IX\_ETH\_DB\_CHECK\_PORT**(portID)

*safety checks to verify whether the port is invalid or uninitialized*

## Enumerations

enum **IxEthDBPortType** {  
    **ETH\_GENERIC**,  
    **ETH\_NPE**  
}

*Port types – currently only Ethernet NPEs are recognized as specific types.*

enum **IxEthDBPortCapability** {  
    **NO\_CAPABILITIES**,  
    **ENTRY\_AGING**  
}

*Port capabilities – used by ixEthAccDatabaseMaintenance to decide whether it should manually age entries or not depending on the port capabilities.*

---

## Detailed Description

IXP425 Ethernet Port Definitions for private MAC learning API.

---

## Define Documentation

```
#define COMPLETE_ETH_PORT_MAP
```

complete set of ports in use

only ports 0, 1 and 2 are in use – sets bit[n] to 1 if port[n] exists

Definition at line **122** of file **IxEthDBPortDefs.h**.

```
#define IX_ETH_DB_CHECK_PORT ( portID )
```

safety checks to verify whether the port is invalid or uninitialized

Definition at line **128** of file **IxEthDBPortDefs.h**.

```
#define IX_ETH_DB_NUMBER_OF_PORTS
```

number of supported ports

Definition at line **107** of file **IxEthDBPortDefs.h**.

```
#define IX_ETH_DB_PORTS_ASSERTION
```

catch invalid port definitions (<2) with a compile–time assertion resulting in a duplicate case error.

Definition at line **114** of file **IxEthDBPortDefs.h**.

---

## Enumeration Type Documentation

```
enum IxEthDBPortCapability
```

Port capabilities – used by ixEthAccDatabaseMaintenance to decide whether it should manually age entries or not depending on the port capabilities.

Ethernet NPEs have aging capabilities, meaning that they will age the entries automatically (by themselves).

***Enumeration values:***

*NO\_CAPABILITIES* no aging capabilities  
*ENTRY\_AGING* aging capabilities present

Definition at line **72** of file **IxEthDBPortDefs.h**.

```
enum IxEthDBPortType
```

Port types – currently only Ethernet NPEs are recognized as specific types.

***Enumeration values:***

*ETH\_GENERIC* generic ethernet port – not supported  
*ETH\_NPE* specific Ethernet NPE

Definition at line **60** of file **IxEthDBPortDefs.h**.

# IXP425 Ethernet Phy Access (IxEthMii) API

ethMii is a library that does provides access to the Ethernet PHYs

## Functions

IX\_STATUS **ixEthMiiPhyScan** (BOOL phyPresent[], UINT32 maxPhyCount)

*Scan the MDIO bus for PHYs This function scans PHY addresses 0 through 31, and sets phyPresent[n] to TRUE if a phy is discovered at address n.*

IX\_STATUS **ixEthMiiPhyConfig** (UINT32 phyAddr, BOOL speed100, BOOL fullDuplex, BOOL autonegotiate)

*Configure a PHY Configure a PHY's speed, duplex and autonegotiation status.*

IX\_STATUS **ixEthMiiPhyLoopbackEnable** (UINT32 phyAddr)

*Enable PHY Loopback in a specific Eth MII port.*

IX\_STATUS **ixEthMiiPhyLoopbackDisable** (UINT32 phyAddr)

*Disable PHY Loopback in a specific Eth MII port.*

IX\_STATUS **ixEthMiiPhyReset** (UINT32 phyAddr)

*Reset a PHY Reset a PHY.*

IX\_STATUS **ixEthMiiLinkStatus** (UINT32 phyAddr, BOOL \*linkUp, BOOL \*speed100, BOOL \*fullDuplex, BOOL \*autoneg)

*Retrieve the current status of a PHY Retrieve the link, speed, duplex and autonegotiation status of a PHY.*

IX\_STATUS **ixEthMiiPhyShow** (UINT32 phyAddr)

*Display information on a specified PHY Display link status, speed, duplex and Auto Negotiation status.*

---

## Detailed Description

ethMii is a library that does provides access to the Ethernet PHYs

---

## Function Documentation

```
ixEthMiiLinkStatus ( UINT32 phyAddr,
                     BOOL * linkUp,
                     BOOL * speed100,
                     BOOL * fullDuplex,
                     BOOL * autoneg
                     )
```

Retrieve the current status of a PHY Retrieve the link, speed, duplex and autonegotiation status of a PHY.

\* – Reentrant – no

- ISR Callable – no

**Precondition:**

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

**Parameters:**

*phyAddr*: the address of the Ethernet PHY (0–31)  
*linkUp* : set to TRUE if the link is up  
*speed100*: set to TRUE indicates 100Mbit/s, FALSE indicates 10Mbit/s  
*fullDuplex*: set to TRUE indicates Full Duplex, FALSE indicates Half Duplex  
*autoneg* : set to TRUE indicates autonegotiation is enabled, FALSE indicates autonegotiation is disabled

**Returns:**

IX\_STATUS  
◇ IX\_SUCCESS  
◇ IX\_FAIL : invalid arguments.

---

```
ixEthMiiPhyConfig ( UINT32 phyAddr,  
                    BOOL   speed100,  
                    BOOL   fullDuplex,  
                    BOOL   autonegotiate  
                    )
```

Configure a PHY Configure a PHY's speed, duplex and autonegotiation status.

\* – Reentrant – no

- ISR Callable – no

**Precondition:**

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

**Parameters:**

*phyAddr*  
*speed100*: set to TRUE for 100Mbit/s operation, FALSE for 10Mbit/s  
*fullDuplex*: set to TRUE for Full Duplex, FALSE for Half Duplex  
*autonegotiate*: set to TRUE to enable autonegotiation

**Returns:**

IX\_STATUS  
◇ IX\_SUCCESS

◇ IX\_FAIL : invalid arguments.

---

**ixEthMiiPhyLoopbackDisable ( UINT32 *phyAddr* )**

Disable PHY Loopback in a specific Eth MII port.

\* – Reentrant – no

- ISR Callable – no

**Parameters:**

*phyAddr[in]* – the address of the Ethernet PHY  
(0–31)

**Returns:**

IX\_STATUS  
◇ IX\_SUCCESS  
◇ IX\_FAIL : invalid arguments.

---

**ixEthMiiPhyLoopbackEnable ( UINT32 *phyAddr* )**

Enable PHY Loopback in a specific Eth MII port.

**Note:**

When PHY Loopback is enabled, frames sent out to the PHY from the IXP425 will be looped back to the IXP425. They will not be transmitted out on the wire.

- Reentrant – no
- ISR Callable – no

**Parameters:**

*phyAddr[in]* – the address of the Ethernet PHY (0–31)

**Returns:**

IX\_STATUS  
◇ IX\_SUCCESS  
◇ IX\_FAIL : invalid arguments.

---

**ixEthMiiPhyReset ( UINT32 *phyAddr* )**

Reset a PHY Reset a PHY.

\* – Reentrant – no

- ISR Callable – no

**Precondition:**

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

**Parameters:**

*phyAddr*: the address of the Ethernet PHY (0–31)

**Returns:**

IX\_STATUS  
 ◇ IX\_SUCCESS  
 ◇ IX\_FAIL : invalid arguments.

---

```
ixEthMiiPhyScan ( BOOL  phyPresent[],
                  UINT32 maxPhyCount
                  )
```

Scan the MDIO bus for PHYs This function scans PHY addresses 0 through 31, and sets phyPresent[n] to TRUE if a phy is discovered at address n.

\* – Reentrant – no

- ISR Callable – no

**Precondition:**

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

**Parameters:**

*phyPresent* : boolean array of IXP425\_ETH\_ACC\_MII\_MAX\_ADDR entries  
*maxPhyCount* : number of PHYs to search for (the scan will stop when the indicated number of PHYs is found).

**Returns:**

IX\_STATUS  
 ◇ IX\_ETH\_ACC\_SUCCESS  
 ◇ IX\_ETH\_ACC\_FAIL : invalid arguments.

---

```
ixEthMiiPhyShow ( UINT32 phyAddr )
```

Display information on a specified PHY Display link status, speed, duplex and Auto Negotiation status.

\* – Reentrant – no

- ISR Callable – no

***Precondition:***

The MAC on Ethernet Port 2 (NPE C) must be initialised, and generating the MDIO clock.

***Parameters:***

*phyAddr*: the address of the Ethernet PHY (0–31)

***Returns:***

IX\_STATUS

◊ IX\_SUCCESS

◊ IX\_FAIL : invalid arguments.

---



# IXP425 Ethernet NPE (IxEthNpe) API

Contains the API for Ethernet NPE.

## Defines

**#define IX\_ETHNPE\_X2P\_NPE\_HALT**

*Request from the XScale client for the NPE to immediately halt all execution and flush any mbufs in its possession.*

**#define IX\_ETHNPE\_X2P\_NPE\_PORT\_DISABLE**

*Request from the XScale client for the NPE to immediately flush any mbufs in its possession.*

**#define IX\_ETHNPE\_X2P\_ELT\_SETPORTADDRESS**

*Indication from the XScale client that the attached Ethernet port's MAC address is equal to the specified value and that the port ID of attached Ethernet port should be set to the specified value.*

**#define IX\_ETHNPE\_X2P\_NPE\_SETMAXSIZEFILTERING1**

*Request from the XScale client for the NPE to update the the maximum frame size per port.*

**#define IX\_ETHNPE\_X2P\_NPE\_SETMAXSIZEFILTERING2**

*Request from the XScale client for the NPE to update the the maximum frame size per port.*

**#define IX\_ETHNPE\_X2P\_ELT\_ACCESSREQUEST**

*Request from the XScale client for the NPE to relinquish control of the Ethernet Learning Tree and write it back to external memory (at the location specified in the last X2P\_ELT\_AccessRelease message).*

**#define IX\_ETHNPE\_X2P\_ELT\_ACCESSRELEASE**

*Indication from the XScale client that it has relinquished control of the Ethernet Learning Tree and has written an updated version of it, with its base node at the specified address (the base node is the empty node immediately preceding the true root node).*

**#define IX\_ETHNPE\_X2P\_ELT\_INSERTADDRESS**

*Indication from the XScale client that the NPE should insert the specified MAC address/Port ID into internal tree.*

**#define IX\_ETHNPE\_X2P\_STATS\_SHOW**

*Request from the XScale client for the current MAC port statistics data to be written to the (empty) statistics structure and the specified location in external memory.*

**#define IX\_ETHNPE\_X2P\_STATS\_RESET**

*Request from the XScale client for the NPE to reset all of its internal MAC port statistics state variables.*

**#define IX\_ETHNPE\_P2X\_NPE\_STATUS**

*Indication from the NPE of its current status.*

```

#define IX_ETHNPE_P2X_ELT_ACKPORTADDRESS
    Indication from the NPE that it has finished processing the previous X2P_ELT_SetPortAddress message.

#define IX_ETHNPE_P2X_NPE_ACKMAXSIZEFILTERING1
    Indication from the NPE that it has finished processing the previous X2P_ELT_SetMaximumFrameSize message.

#define IX_ETHNPE_P2X_NPE_ACKMAXSIZEFILTERING2
    Indication from the NPE that it has finished processing the previous X2P_ELT_SetMaximumFrameSize message.

#define IX_ETHNPE_P2X_ELT_ACCESSGRANT
    Indication from the NPE that it relinquished control of the Ethernet Learning Tree and has written it back to external memory at the specified base address.

#define IX_ETHNPE_P2X_ELT_BALANCEREQUEST
    Request from the NPE for the XScale client to insert the specified MAC address into the Ethernet Learning Tree and rebalance it (the NPE has run out of depth while attempting to insert the source MAC address itself).

#define IX_ETHNPE_P2X_ELT_NEWADDRESS
    Indication from the NPE that it has just learned (i.e. inserted into its internal tree) the specified new MAC address.

#define IX_ETHNPE_P2X_ELT_INSERTADDRESSACK
    Indication from the NPE that it has successfully enqueued (to the learning process) the MAC address from the previous X2P_ELT_Insert_Address message.

#define IX_ETHNPE_P2X_ELT_INSERTADDRESSNACK
    Indication from the NPE that it is unable to enqueue (to the learning process) the MAC address from the previous X2P_ELT_Insert_Address message.

#define IX_ETHNPE_P2X_STATS_REPORT
    Indication from the NPE that the current MAC port statistics are available in the specified buffer.

#define IX_ETHNPE_P2X_STATS_CLEAR_REPORT
    Indication from the NPE that the current MAC port statistics are cleared.

#define IX_ETHNPE_P2X_NPE_PORT_DISABLE
    Response to a IX_ETHNPE_X2P_NPE_PORT_DISABLE.

#define MASK(hi, lo)
    Macro for mask.

#define BITS(x, hi, lo)
    Macro for bits.

#define IX_ETHNPE_QM_Q_RXENET_LENGTH_MASK
    QMgr Queue LENGTH field mask.

```

```

#define IX_ETHNPE_QM_Q_FIELD_FLAG_R
    QMgr Queue FLAG field right boundary.

#define IX_ETHNPE_QM_Q_FIELD_FLAG_MASK
    QMgr Queue FLAG field mask.

#define IX_ETHNPE_QM_Q_FIELD_NPEID_L
    QMgr Queue NPE ID field left boundary.

#define IX_ETHNPE_QM_Q_FIELD_NPEID_R
    QMgr Queue NPE ID field right boundary.

#define IX_ETHNPE_QM_Q_FIELD_PORTID_L
    QMgr Queue Port ID field left boundary.

#define IX_ETHNPE_QM_Q_FIELD_PORTID_R
    QMgr Queue Port ID field right boundary.

#define IX_ETHNPE_QM_Q_FIELD_PRIOR_L
    QMgr Queue Priority field left boundary.

#define IX_ETHNPE_QM_Q_FIELD_PRIOR_R
    QMgr Queue Priority field right boundary.

#define IX_ETHNPE_QM_Q_FIELD_ADDR_L
    QMgr Queue Address field left boundary.

#define IX_ETHNPE_QM_Q_FIELD_ADDR_R
    QMgr Queue Address field right boundary.

#define IX_ETHNPE_QM_Q_FREEENET_ADDR_MASK
    Macro to mask the Address field of the FreeEnet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_RXENET_NPEID_MASK
    Macro to mask the NPE ID field of the RxEnet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_RXENET_PORTID_MASK
    Macro to mask the Port ID field of the Queue Manager RxEnet Queue entry.

#define IX_ETHNPE_QM_Q_RXENET_ADDR_MASK
    Macro to mask the Mbuf Address field of the RxEnet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_TXENET_PRIOR_MASK
    Macro to mask the Priority field of the TxEnet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_TXENET_ADDR_MASK
    Macro to mask the Mbuf Address field of the TxEnet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_TXENETDONE_NPEID_MASK
    Macro to mask the NPE ID field of the TxEnetDone Queue Manager Entry.

```

```

#define IX_ETHNPE_QM_Q_TXENETDONE_ADDR_MASK
    Macro to mask the Mbuf Address field of the TxEnetDone Queue Manager Entry.

#define IX_ETHNPE_QM_Q_FREEENET_ADDR_VAL(x)
    Extraction macro for Address field of FreeNet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_RXENET_NPEID_VAL(x)
    Extraction macro for NPE ID field of RxEnet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_RXENET_PORTID_VAL(x)
    Extraction macro for Port ID field of RxEnet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_RXENET_ADDR_VAL(x)
    Extraction macro for Address field of RxEnet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_TXENET_PRIOR_VAL(x)
    Extraction macro for Priority field of TxEnet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_TXENET_ADDR_VAL(x)
    Extraction macro for Address field of Queue Manager TxEnet Queue Manager Entry.

#define IX_ETHNPE_QM_Q_TXENETDONE_NPEID_VAL(x)
    Extraction macro for NPE ID field of TxEnetDone Queue Manager Entry.

#define IX_ETHNPE_QM_Q_TXENETDONE_ADDR_VAL(x)
    Extraction macro for Address field of TxEnetDone Queue Manager Entry.

#define IX_ETHNPE_QM_Q_FREEENET_ENTRY(addr)
    Queue entry construction macros for FreeNet Queue Manager.

#define IX_ETHNPE_QM_Q_RXENET_ENTRY(addr, id, prt)
    Queue entry construction macros for RxEnet Queue Manager.

#define IX_ETHNPE_QM_Q_TXENET_ENTRY(addr, pri)
    Queue entry construction macros for TxEnet Queue Manager.

#define IX_ETHNPE_QM_Q_TXENETDONE_ENTRY(addr, id)
    Queue entry construction macros for TxEnetDone Queue Manager.

#define IX_ETHNPE_ACC_RXFREE_BUFFER_LENGTH_MIN
    Macro to check the minimum length of a rx free buffer.

#define IX_ETHNPE_ACC_RXFREE_BUFFER_LENGTH_MASK
    Mask to apply to the mbuf length before submitting it to the NPE (the NPE handles only rx free mbufs which are multiple of 64).

#define IX_ETHNPE_ACC_RXFREE_BUFFER_ROUND_UP(size)
    Round up to apply to the mbuf length before submitting it to the NPE (the NPE handles only rx free mbufs which are multiple of 64).

#define IX_ETHNPE_ACC_FRAME_LENGTH_MAX

```

*maximum mbuf length supported by the NPE*

```
#define IX_ETHNPE_ACC_FRAME_LENGTH_DEFAULT  
default mbuf length supported by the NPE
```

---

## Detailed Description

Contains the API for Ethernet NPE.

---

## Define Documentation

```
#define BITS ( x,  
             hi,  
             lo )
```

Macro for bits.

Definition at line **283** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_ACC_FRAME_LENGTH_DEFAULT
```

*default mbuf length supported by the NPE*

*See also:*

**IX\_ETHNPE\_ACC\_FRAME\_LENGTH\_DEFAULT**

Definition at line **634** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_ACC_FRAME_LENGTH_MAX
```

*maximum mbuf length supported by the NPE*

*See also:*

**IX\_ETHNPE\_ACC\_FRAME\_LENGTH\_MAX**

Definition at line **625** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_ACC_RXFREE_BUFFER_LENGTH_MASK
```

Mask to apply to the mbuf length before submitting it to the NPE (the NPE handles only rx free mbufs which are multiple of 64).

*See also:*

## **IX\_ETHNPE\_ACC\_RXFREE\_BUFFER\_LENGTH\_MASK**

Definition at line **607** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_ACC_RXFREE_BUFFER_LENGTH_MIN
```

Macro to check the minimum length of a rx free buffer.

Definition at line **597** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_ACC_RXFREE_BUFFER_ROUND_UP ( size )
```

Round up to apply to the mbuf length before submitting it to the NPE (the NPE handles only rx free mbufs which are multiple of 64).

Definition at line **616** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_ELT_ACCESSGRANT
```

Indication from the NPE that it relinquished control of the Ethernet Learning Tree and has written it back to external memory at the specified base address.

Definition at line **200** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_ELT_ACKPORTADDRESS
```

Indication from the NPE that it has finished processing the previous X2P\_ELT\_SetPortAddress message.

Definition at line **175** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_ELT_BALANCEREQUEST
```

Request from the NPE for the XScale client to insert the specified MAC address into the Ethernet Learning Tree and rebalance it (the NPE has run out of depth while attempting to insert the source MAC address itself).

A P2X\_ELT\_AccessGrant message is implied (i.e. the NPE will have relinquished control of the tree and written it back to external memory prior to issuing this message).

Definition at line **213** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_ELT_INSERTADDRESSACK
```

Indication from the NPE that it has successfully enqueued (to the learning process) the MAC address from the previous X2P\_ELT\_Insert\_Address message.

Definition at line **229** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_ELT_INSERTADDRESSNACK
```

Indication from the NPE that it is unable to enqueue (to the learning process) the MAC address from the previous X2P\_ELT\_Insert\_Address message.

Definition at line **237** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_ELT_NEWADDRESS
```

Indication from the NPE that it has just learned (i.e. inserted into its internal tree) the specified new MAC address.

Definition at line **221** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_NPE_ACKMAXSIZEFILTERING1
```

Indication from the NPE that it has finished processing the previous X2P\_ELT\_SetMaximumFrameSize message.

Definition at line **183** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_NPE_ACKMAXSIZEFILTERING2
```

Indication from the NPE that it has finished processing the previous X2P\_ELT\_SetMaximumFrameSize message.

Definition at line **191** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_NPE_PORT_DISABLE
```

Response to a IX\_ETHNPE\_X2P\_NPE\_PORT\_DISABLE.

Indication from the NPE that mbufs held are released (msg word1 is null) or there is still pending traffic (msg word1 is not null)

Definition at line **265** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_NPE_STATUS
```

Indication from the NPE of its current status.

Definition at line **167** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_STATS_CLEAR_REPORT
```

Indication from the NPE that the current MAC port statistics are cleared.

Definition at line **252** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_P2X_STATS_REPORT
```

Indication from the NPE that the current MAC port statistics are available in the specified buffer.

Definition at line **245** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FIELD_ADDR_L
```

QMgr Queue Address field left boundary.

Definition at line **360** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FIELD_ADDR_R
```

QMgr Queue Address field right boundary.

Definition at line **367** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FIELD_FLAG_MASK
```

QMgr Queue FLAG field mask.

Multicast bit : BIT(4) Broadcast bit : BIT(5) IP bit : BIT(6) (linux only)

Definition at line **310** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FIELD_FLAG_R
```

QMgr Queue FLAG field right boundary.

Definition at line **297** of file **IxEthNpe.h**.



```
#define IX_ETHNPE_QM_Q_FIELD_NPEID_L
```

QMgr Queue NPE ID field left boundary.

Definition at line **318** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FIELD_NPEID_R
```

QMgr Queue NPE ID field right boundary.

Definition at line **325** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FIELD_PORTID_L
```

QMgr Queue Port ID field left boundary.

Definition at line **332** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FIELD_PORTID_R
```

QMgr Queue Port ID field right boundary.

Definition at line **339** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FIELD_PRIOR_L
```

QMgr Queue Priority field left boundary.

Definition at line **346** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FIELD_PRIOR_R
```

QMgr Queue Priority field right boundary.

Definition at line **353** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FREEENET_ADDR_MASK
```

Macro to mask the Address field of the FreeEnet Queue Manager Entry.

Definition at line **378** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FREEENET_ADDR_VAL ( x )
```

Extraction macro for Address field of FreeNet Queue Manager Entry.

Pointer to an mbuf buffer descriptor

Definition at line **466** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_FREEENET_ENTRY ( addr )
```

Queue entry construction macros for FreeNet Queue Manager.

Definition at line **563** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_RXENET_ADDR_MASK
```

Macro to mask the Mbuf Address field of the RxEnet Queue Manager Entry.

Definition at line **411** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_RXENET_ADDR_VAL ( x )
```

Extraction macro for Address field of RxEnet Queue Manager Entry.

Pointer to an mbuf buffer descriptor

Definition at line **502** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_RXENET_ENTRY ( addr,  
                                     id,  
                                     prt )
```

Queue entry construction macros for RxEnet Queue Manager.

Definition at line **570** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_RXENET_LENGTH_MASK
```

QMgr Queue LENGTH field mask.

Definition at line **290** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_RXENET_NPEID_MASK
```

Macro to mask the NPE ID field of the RxEnet Queue Manager Entry.

Definition at line **392** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_RXENET_NPEID_VAL ( x )
```

Extraction macro for NPE ID field of RxEnet Queue Manager Entry.

Set to 0 for entries originating from the Eth0 NPE; Set to 1 for entries originating from the Eth1 NPE.

Definition at line **477** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_RXENET_PORTID_MASK
```

Macro to mask the Port ID field of the Queue Manager RxEnet Queue entry.

Definition at line **404** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_RXENET_PORTID_VAL ( x )
```

Extraction macro for Port ID field of RxEnet Queue Manager Entry.

0–5: Assignable (by the XScale client) to any of the physical ports. 6: It is reserved 7: Indication that the NPE did not find the associated frame's destination MAC address within its internal filtering database.

Definition at line **491** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_TXENET_ADDR_MASK
```

Macro to mask the Mbuf Address field of the TxEnet Queue Manager Entry.

Definition at line **429** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_TXENET_ADDR_VAL ( x )
```

Extraction macro for Address field of Queue Manager TxEnet Queue Manager Entry.

Pointer to an mbuf buffer descriptor

Definition at line **525** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_TXENET_ENTRY ( addr,  
                                     pri  )
```

Queue entry construction macros for TxEnet Queue Manager.

Definition at line **578** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_TXENET_PRIOR_MASK
```

Macro to mask the Priority field of the TxEnet Queue Manager Entry.

Definition at line **420** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_TXENET_PRIOR_VAL ( x )
```

Extraction macro for Priority field of TxEnet Queue Manager Entry.

Priority of the packet (as described in IEEE 802.1D). This field is cleared upon return from the Ethernet NPE to the TxEnetDone queue.

Definition at line **513** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_TXENETDONE_ADDR_MASK
```

Macro to mask the Mbuf Address field of the TxEnetDone Queue Manager Entry.

Definition at line **451** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_TXENETDONE_ADDR_VAL ( x )
```

Extraction macro for Address field of TxEnetDone Queue Manager Entry.

Pointer to an mbuf buffer descriptor

Definition at line **547** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_TXENETDONE_ENTRY ( addr,  
                                         id  )
```

Queue entry construction macros for TxEnetDone Queue Manager.

Definition at line **585** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_TXENETDONE_NPEID_MASK
```

Macro to mask the NPE ID field of the TxEnetDone Queue Manager Entry.

Definition at line **443** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_QM_Q_TXENETDONE_NPEID_VAL ( x )
```

Extraction macro for NPE ID field of TxEnetDone Queue Manager Entry.

Set to 0 for entries originating from the Eth0 NPE; set to 1 for entries originating from the Eth1 NPE.

Definition at line **536** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_X2P_ELT_ACCESSRELEASE
```

Indication from the XScale client that it has relinquished control of the Ethernet Learning Tree and has written an updated version of it, with its base node at the specified address (the base node is the empty node immediately preceding the true root node).

The tree will remain at the same location until the next X2P\_ELT\_AccessRelease message.

Definition at line **127** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_X2P_ELT_ACCESSREQUEST
```

Request from the XScale client for the NPE to relinquish control of the Ethernet Learning Tree and write it back to external memory (at the location specified in the last X2P\_ELT\_AccessRelease message).

Definition at line **114** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_X2P_ELT_INSERTADDRESS
```

Indication from the XScale client that the NPE should insert the specified MAC address/Port ID into internal tree.

Definition at line **135** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_X2P_ELT_SETPORTADDRESS
```

Indication from the XScale client that the attached Ethernet port's MAC address is equal to the specified value and that the port ID of attached Ethernet port should be set to the specified value.

Definition at line **89** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_X2P_NPE_HALT
```

Request from the XScale client for the NPE to immediately halt all execution and flush any mbufs in its possession.

Any free mbuf held by the NPE receive process is flushed to the RxEnet queue. Transmit path mbufs (those in the Priority Queue and any one currently in the process of transmission) are immediately flushed to the TxEnetDone queue.

Definition at line **67** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_X2P_NPE_PORT_DISABLE
```

Request from the XScale client for the NPE to immediately flush any mbufs in its possession.

Any free mbuf held by the NPE receive process is flushed to the RxEnet queue. Transmit path mbufs (those in the Priority Queue and any one currently in the process of transmission) are immediately flushed to the TxEnetDone queue.

Definition at line **80** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_X2P_NPE_SETMAXSIZEFILTERING1
```

Request from the XScale client for the NPE to update the the maximum frame size per port.

Definition at line **97** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_X2P_NPE_SETMAXSIZEFILTERING2
```

Request from the XScale client for the NPE to update the the maximum frame size per port.

Definition at line **105** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_X2P_STATS_RESET
```

Request from the XScale client for the NPE to reset all of its internal MAC port statistics state variables.

As a side effect, this message entails an implicit request that the NPE write the current MAC port statistics into the MAC statistics structure at the specified location in external memory.

Definition at line **156** of file **IxEthNpe.h**.

```
#define IX_ETHNPE_X2P_STATS_SHOW
```

Request from the XScale client for the current MAC port statistics data to be written to the (empty) statistics structure and the specified location in external memory.

Definition at line **144** of file **IxEthNpe.h**.

```
#define MASK ( hi,  
              lo )
```

Macro for mask.

Definition at line **276** of file **IxEthNpe.h**.

# IXP425 Feature Control (IxFeatureCtrl) API

The Public API for the IXP425 Feature Control.

## Modules

Software  
Configuration for  
Access Component

*This section describes software configuration in access component. The configuration can be changed at run-time. **ixFeatureCtrlSwConfigurationCheck( )** will be used across applicable access component to check the configuration. **ixFeatureCtrlSwConfigurationWrite( )** is used to write the software configuration.*

## Defines

```
#define IX_FEATURE_CTRL_COMPONENT_DISABLED  
    Hardware Component is disabled/unavailable. Return status by  
    ixFeatureCtrlComponentCheck().  
  
#define IX_FEATURE_CTRL_COMPONENT_ENABLED  
    Hardware Component is available. Return status by  
    ixFeatureCtrlComponentCheck().  
  
#define IX_FEATURE_CTRL_SILICON_TYPE_A0  
    This is the value of A0 Silicon in product ID.  
  
#define IX_FEATURE_CTRL_SILICON_TYPE_B0  
    This is the value of B0 Silicon in product ID.  
  
#define IX_FEATURE_CTRL_SILICON_STEPPING_MASK  
    This is the mask of silicon stepping in product ID.  
  
#define IX_FEATURE_CTRL_XSCALE_FREQ_533  
    This is the value of 533MHz XScale Core in product ID.  
  
#define IX_FEATURE_CTRL_XSCALE_FREQ_400  
    This is the value of 400MHz XScale Core in product ID.  
  
#define IX_FEATURE_CTRL_XSCALE_FREQ_266  
    This is the value of 266MHz XScale Core in product ID.  
  
#define IX_FEATURE_CTRL_XSCALE_FREQ_MASK  
    This is the mask of XScale Core in product ID.  
  
#define IX_FEATURECTRL_REG_LOC_RCOMP
```



*The bit location for RComp Circuitry.*

**#define IX\_FEATURECTRL\_REG\_LOC\_USB**

*The bit location for USB Controller.*

**#define IX\_FEATURECTRL\_REG\_LOC\_HASH**

*The bit location for Hashing Coprocessor.*

**#define IX\_FEATURECTRL\_REG\_LOC\_AES**

*The bit location for AES Coprocessor.*

**#define IX\_FEATURECTRL\_REG\_LOC\_DES**

*The bit location for DES Coprocessor.*

**#define IX\_FEATURECTRL\_REG\_LOC\_HDLC**

*The bit location for HDLC Coprocessor.*

**#define IX\_FEATURECTRL\_REG\_LOC\_AAL**

*The bit location for AAL Coprocessor.*

**#define IX\_FEATURECTRL\_REG\_LOC\_HSS**

*The bit location for HSS Coprocessor.*

**#define IX\_FEATURECTRL\_REG\_LOC\_UTOPIA**

*The bit location for UTOPIA Coprocessor.*

**#define IX\_FEATURECTRL\_REG\_LOC\_ETH0**

*The bit location for Ethernet 0 Coprocessor.*

**#define IX\_FEATURECTRL\_REG\_LOC\_ETH1**

*The bit location for Ethernet 1 Coprocessor.*

**#define IX\_FEATURECTRL\_REG\_LOC\_NPEA**

*The bit location for NPE A.*

**#define IX\_FEATURECTRL\_REG\_LOC\_NPEB**

*The bit location for NPE B.*

**#define IX\_FEATURECTRL\_REG\_LOC\_NPEC**

*The bit location for NPE C.*

**#define IX\_FEATURECTRL\_REG\_LOC\_PCI**

*The bit location for PCI Controller.*

**#define IX\_FEATURECTRL\_REG\_LOC\_UTOPIA\_PHY\_LIMIT**

*The bit location for Utopia PHY Limit Status.*

**#define IX\_FEATURECTRL\_RCOMP**

*The Component Name for RCOMP Circuitry. The name will be used for  
**ixFeatureCtrlComponentCheck()**.*

```

#define IX_FEATURECTRL_USB
    The Component Name for USB Controller. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_HASH
    The Component Name for Hashing Coprocessor. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_AES
    The Component Name for AES Coprocessor. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_DES
    The Component Name for DES Coprocessor. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_HDLC
    The Component Name for HDLC Coprocessor. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_AAL
    The Component Name for AAL Coprocessor. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_HSS
    The Component Name for HSS Coprocessor. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_UTOPIA
    The Component Name for Utopia Coprocessor. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_ETH0
    The Component Name for Ethernet 0 Coprocessor. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_ETH1
    The Component Name for Ethernet 1 Coprocessor. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_NPEA
    The Component Name for NPE A. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_NPEB
    The Component Name for NPE B. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_NPEC
    The Component Name for NPE C. The name will be used for
    ixFeatureCtrlComponentCheck().

```

```
#define IX_FEATURECTRL_PCI
    The Component Name for PCI Controller. The name will be used for
    ixFeatureCtrlComponentCheck().

#define IX_FEATURECTRL_REG_UTOPIA_32PHY
    Maximum UTOPIA PHY available to IXP425 is 32.

#define IX_FEATURECTRL_REG_UTOPIA_16PHY
    Maximum UTOPIA PHY available to IXP425 is 16.

#define IX_FEATURECTRL_REG_UTOPIA_8PHY
    Maximum UTOPIA PHY available to IXP425 is 8.

#define IX_FEATURECTRL_REG_UTOPIA_4PHY
    Maximum UTOPIA PHY available to IXP425 is 4.
```

## Typedefs

```
typedef UINT32 IxFeatureCtrlReg
    Feature Control Register that contains hardware components' availability
    information.

typedef UINT32 IxFeatureCtrlProductId
    Product ID of Silicon that contains Silicon Stepping and Maximum XScale Core
    Frequency information.

typedef UINT32 IxFeatureCtrlComponentType
    The component type used for ixFeatureCtrlComponentCheck().
```

## Functions

```
IxFeatureCtrlReg ixFeatureCtrlRead (void)
    This function reads out the CURRENT value of Feature Control Register. The
    current value may not be the same as that of the hardware component availability.

IxFeatureCtrlReg ixFeatureCtrlHwCapabilityRead (void)
    This function reads out the hardware capability of a silicon type as defined in
    feature control register. This value is different from that returned by
    ixFeatureCtrlRead() because this function returns the actual hardware component
    availability.

void ixFeatureCtrlWrite (IxFeatureCtrlReg expUnitReg)
    This function write the value stored in IxFeatureCtrlReg expUnitReg to the Feature
    Control Register.

IX_STATUS ixFeatureCtrlComponentCheck (IxFeatureCtrlComponentType
    componentType)
```

*This function will check the availability of hardware component specified as componentType value.*

**IxFeatureCtrlProductId** **IxFeatureCtrlProductIdRead** (void)

*This function will return IXP425 product ID i.e. CP15, Register 0.*

IX\_STATUS **IxFeatureCtrlSwConfigurationCheck** (**IxFeatureCtrlSwConfig** swConfigType)

*This function checks whether the specified software configuration is enabled or disabled.*

void **IxFeatureCtrlSwConfigurationWrite** (**IxFeatureCtrlSwConfig** swConfigType, BOOL enabled)

*This function enable/disable the specified software configuration.*

void **IxFeatureCtrlIxp400SwVersionShow** (void)

*This function shows the current software release information for IXP400.*

---

## Detailed Description

The Public API for the IXP425 Feature Control.

---

## Define Documentation

```
#define IX_FEATURE_CTRL_COMPONENT_DISABLED
```

Hardware Component is disabled/unavailable. Return status by **IxFeatureCtrlComponentCheck()**.

Definition at line **85** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURE_CTRL_COMPONENT_ENABLED
```

Hardware Component is available. Return status by **IxFeatureCtrlComponentCheck()**.

Definition at line **95** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURE_CTRL_SILICON_STEPPING_MASK
```

This is the mask of silicon stepping in product ID.

Definition at line **149** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURE_CTRL_SILICON_TYPE_A0
```

This is the value of A0 Silicon in product ID.

Definition at line **131** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURE_CTRL_SILICON_TYPE_B0
```

This is the value of B0 Silicon in product ID.

Definition at line **140** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURE_CTRL_XSCALE_FREQ_266
```

This is the value of 266MHz XScale Core in product ID.

Definition at line **176** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURE_CTRL_XSCALE_FREQ_400
```

This is the value of 400MHz XScale Core in product ID.

Definition at line **167** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURE_CTRL_XSCALE_FREQ_533
```

This is the value of 533MHz XScale Core in product ID.

Definition at line **158** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURE_CTRL_XSCALE_FREQ_MASK
```

This is the mask of XScale Core in product ID.

Definition at line **185** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_AAL
```

The Component Name for AAL Coprocessor. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **444** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_AES
```

The Component Name for AES Coprocessor. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **414** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_DES
```

The Component Name for DES Coprocessor. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **424** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_ETH0
```

The Component Name for Ethernet 0 Coprocessor. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **474** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_ETH1
```

The Component Name for Ethernet 1 Coprocessor. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **484** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_HASH
```

The Component Name for Hashing Coprocessor. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **404** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_HDLC
```

The Component Name for HDLC Coprocessor. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **434** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_HSS
```

The Component Name for HSS Coprocessor. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **454** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_NPEA
```

The Component Name for NPE A. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **494** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_NPEB
```

The Component Name for NPE B. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **504** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_NPEC
```

The Component Name for NPE C. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **514** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_PCI
```

The Component Name for PCI Controller. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **524** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_RCOMP
```

The Component Name for RCOMP Circuitry. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **384** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_AAL
```

The bit location for AAL Coprocessor.

Definition at line **288** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_AES
```

The bit location for AES Coprocessor.

Definition at line **261** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_DES
```

The bit location for DES Coprocessor.

Definition at line **270** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_ETH0
```

The bit location for Ethernet 0 Coprocessor.

Definition at line **315** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_ETH1
```

The bit location for Ethernet 1 Coprocessor.

Definition at line **324** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_HASH
```

The bit location for Hashing Coprocessor.

Definition at line **252** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_HDLC
```

The bit location for HDLC Coprocessor.

Definition at line **279** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_HSS
```

The bit location for HSS Coprocessor.



Definition at line **297** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_NPEA
```

The bit location for NPE A.

Definition at line **333** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_NPEB
```

The bit location for NPE B.

Definition at line **342** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_NPEC
```

The bit location for NPE C.

Definition at line **351** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_PCI
```

The bit location for PCI Controller.

Definition at line **360** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_RCOMP
```

The bit location for RComp Circuitry.

Definition at line **234** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_USB
```

The bit location for USB Controller.

Definition at line **243** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_UTOPIA
```

The bit location for UTOPIA Coprocessor.

Definition at line **306** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_LOC_UTOPIA_PHY_LIMIT
```

The bit location for Utopia PHY Limit Status.

Definition at line **369** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_UTOPIA_16PHY
```

Maximum UTOPIA PHY available to IXP425 is 16.

Definition at line **544** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_UTOPIA_32PHY
```

Maximum UTOPIA PHY available to IXP425 is 32.

Definition at line **534** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_UTOPIA_4PHY
```

Maximum UTOPIA PHY available to IXP425 is 4.

Definition at line **564** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_REG_UTOPIA_8PHY
```

Maximum UTOPIA PHY available to IXP425 is 8.

Definition at line **554** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_USB
```

The Component Name for USB Controller. The name will be used for **ixFeatureCtrlComponentCheck()**.

Definition at line **394** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURECTRL_UTOPIA
```

The Component Name for Utopia Coprocessor. The name will be used for

**ixFeatureCtrlComponentCheck()**.

Definition at line **464** of file **IxFeatureCtrl.h**.

---

## Typedef Documentation

### IxFeatureCtrlComponentType

The component type used for **ixFeatureCtrlComponentCheck()**.

Definition at line **654** of file **IxFeatureCtrl.h**.

### IxFeatureCtrlProductId

Product ID of Silicon that contains Silicon Stepping and Maximum XScale Core Frequency information.

Definition at line **644** of file **IxFeatureCtrl.h**.

### IxFeatureCtrlReg

Feature Control Register that contains hardware components' availability information.

Definition at line **634** of file **IxFeatureCtrl.h**.

---

## Function Documentation

**IX\_STATUS** ixFeatureCtrlComponentCheck ( **IxFeatureCtrlComponentType** *componentType* )

This function will check the availability of hardware component specified as *componentType* value.

Usage Example:

- if(IX\_FEATURE\_CTRL\_COMPONENT\_DISABLED != ixFeatureCtrlComponentCheck(IX\_FEATURECTRL\_ETH0))
- if(IX\_FEATURE\_CTRL\_COMPONENT\_ENABLED == ixFeatureCtrlComponentCheck(IX\_FEATURECTRL\_PCI))

This function is typically called during component initialization time.

#### **Parameters:**

*IxFeatureCtrlComponentType* (in)*componentType* – the type of a component as defined above as IX\_FEATURECTRL\_XXX (Exp: IX\_FEATURECTRL\_PCI, IX\_FEATURECTRL\_ETH0)

**Returns:**

- ◇ IX\_FEATURE\_CTRL\_COMPONENT\_ENABLED if component is available
- ◇ IX\_FEATURE\_CTRL\_COMPONENT\_DISABLED if component is unavailable

**IxFeatureCtrlReg** ixFeatureCtrlHwCapabilityRead ( void )

This function reads out the hardware capability of a silicon type as defined in feature control register. This value is different from that returned by **ixFeatureCtrlRead()** because this function returns the actual hardware component availability.

The bit location of each hardware component is defined above. A value of '1' in bit means the hardware component is not available. A value of '0' means the hardware component is available.

**Returns:**

- ◇ IxFeatureCtrlReg – the hardware capability of IXP425.

**Warning:**

- ◇ This function must not be called when IXP425 is running as the result is undefined.

**void** ixFeatureCtrlIxp400SwVersionShow ( void )

This function shows the current software release information for IXP400.

**Returns:**

none

**IxFeatureCtrlProductId** ixFeatureCtrlProductIdRead ( void )

This function will return IXP425 product ID i.e. CP15, Register 0.

**Returns:**

- ◇ IxFeatureCtrlProductId – the value of product ID.

**IxFeatureCtrlReg** ixFeatureCtrlRead ( void )

This function reads out the CURRENT value of Feature Control Register. The current value may not be the same as that of the hardware component availability.

The bit location of each hardware component is defined above. A value of '1' in bit means the hardware component is not available. A value of '0' means the hardware component is available.

**Returns:**

- ◇ IxFeatureCtrlReg – the current value of IXP425 Feature Control Register

```
IX_STATUS ixFeatureCtrlSwConfigurationCheck ( IxFeatureCtrlSwConfig swConfigType )
```

This function checks whether the specified software configuration is enabled or disabled.

Usage Example:

- if(IX\_FEATURE\_CTRL\_SWCONFIG\_DISABLED !=  
    ixFeatureCtrlSwConfigurationCheck(IX\_FEATURECTRL\_ETH\_LEARNING))
- if(IX\_FEATURE\_CTRL\_SWCONFIG\_ENABLED ==  
    ixFeatureCtrlSwConfigurationCheck(IX\_FEATURECTRL\_ETH\_LEARNING))

This function is typically called during access component initialization time.

**Parameters:**

*IxFeatureCtrlSwConfig* (in)swConfigType – the type of a software configuration defined in IxFeatureCtrlSwConfig enumeration.

**Returns:**

- ◇ IX\_FEATURE\_CTRL\_SWCONFIG\_ENABLED if software configuration is enabled.
- ◇ IX\_FEATURE\_CTRL\_SWCONFIG\_DISABLED if software configuration is disabled.

```
void ixFeatureCtrlSwConfigurationWrite ( IxFeatureCtrlSwConfig swConfigType,  
                                         BOOL enabled  
                                         )
```

This function enable/disable the specified software configuration.

Usage Example:

- ixFeatureCtrlSwConfigurationWrite(IX\_FEATURECTRL\_ETH\_LEARNING, TRUE) is used to enable Ethernet Learning Feature
- ixFeatureCtrlSwConfigurationWrite(IX\_FEATURECTRL\_ETH\_LEARNING, FALSE) is used to disable Ethernet Learning Feature

**Parameters:**

*IxFeatureCtrlSwConfig* (in)swConfigType – the type of a software configuration defined in IxFeatureCtrlSwConfig enumeration.

*BOOL* (in)enabled – To enable(TRUE) / disable (FALSE) the specified software configuration.

**Returns:**

none

```
void ixFeatureCtrlWrite ( IxFeatureCtrlReg expUnitReg )
```

This function write the value stored in IxFeatureCtrlReg expUnitReg to the Feature Control Register.

The bit location of each hardware component is defined above. The write is only effective on available hardware components. Writing '1' in a bit will software disable the respective hardware component. A '0' will mean that the hardware component will remain to be operable.

***Parameters:***

*IxFeatureCtrlReg* (in)expUnitReg – The value to be written to feature control register.

***Returns:***

none

# Software Configuration for Access Component

## [IXP425 Feature Control (IxFeatureCtrl) API]

This section describes software configuration in access component. The configuration can be changed at run-time. **ixFeatureCtrlSwConfigurationCheck()** will be used across applicable access component to check the configuration. **ixFeatureCtrlSwConfigurationWrite()** is used to write the software configuration.

### Defines

```
#define IX_FEATURE_CTRL_SWCONFIG_DISABLED
    Software configuration is disabled.
```

```
#define IX_FEATURE_CTRL_SWCONFIG_ENABLED
    Software configuration is enabled.
```

### Enumerations

```
enum IxFeatureCtrlSwConfig {
    IX_FEATURECTRL_ETH_LEARNING,
    IX_FEATURECTRL_SWCONFIG_MAX
}
    Enumeration for software configuration in access components.
```

---

## Detailed Description

This section describes software configuration in access component. The configuration can be changed at run-time. **ixFeatureCtrlSwConfigurationCheck()** will be used across applicable access component to check the configuration. **ixFeatureCtrlSwConfigurationWrite()** is used to write the software configuration.

#### *Note:*

**All software configurations are default to be enabled.**

---

## Define Documentation

```
#define IX_FEATURE_CTRL_SWCONFIG_DISABLED
```

Software configuration is disabled.

Definition at line **588** of file **IxFeatureCtrl.h**.

```
#define IX_FEATURE_CTRL_SWCONFIG_ENABLED
```

Software configuration is enabled.

Definition at line **598** of file **IxFeatureCtrl.h**.

---

## Enumeration Type Documentation

```
enum IxFeatureCtrlSwConfig
```

Enumeration for software configuration in access components.

***Enumeration values:***

*IX\_FEATURECTRL\_ETH\_LEARNING* EthDB Learning Feature.

*IX\_FEATURECTRL\_SWCONFIG\_MAX* Maximum boudary for  
IxFeatureCtrlSwConfig.

Definition at line **611** of file **IxFeatureCtrl.h**.



# IXP425 HSS Access (IxHssAcc) API

The public API for the IXP425 HssAccess component.

## Data Structures

- struct **IxHssAccConfigParams**  
*Structure containing HSS configuration parameters.*
- struct **IxHssAccHdlcMode**  
*This structure contains 56Kbps, HDLC-mode configuration parameters.*
- struct **IxHssAccPktHdlcFraming**  
*This structure contains information required by the NPE to configure the HDLC co-processor.*
- struct **IxHssAccPortConfig**  
*Structure containing HSS port configuration parameters.*

## Defines

- #define **IX\_HSSACC\_TSLOTS\_PER\_HSS\_PORT**  
*The max number of TDM timeslots supported per HSS port –  $4EI's = 32 \times 4 = 128$ .*
- #define **IX\_HSSACC\_PARAM\_ERR**  
*HssAccess function return value for a parameter error.*
- #define **IX\_HSSACC\_RESOURCE\_ERR**  
*HssAccess function return value for a resource error.*
- #define **IX\_HSSACC\_PKT\_DISCONNECTING**  
*Indicates that a disconnect call is progressing and will disconnect soon.*
- #define **IX\_HSSACC\_Q\_WRITE\_OVERFLOW**  
*Indicates that an attempt to Tx or to replenish an RxFree Q failed due to Q overflow.*
- #define **IX\_HSSACC\_NO\_ERROR**  
*HSS port no error present.*
- #define **IX\_HSSACC\_TX\_FRM\_SYNC\_ERR**  
*HSS port TX Frame Sync error.*
- #define **IX\_HSSACC\_TX\_OVER\_RUN\_ERR**  
*HSS port TX over-run error.*
- #define **IX\_HSSACC\_CHANNELISED\_SW\_TX\_ERR**

*NPE software error in channelised TX.*

**#define IX\_HSSACC\_PACKETISED\_SW\_TX\_ERR**  
*NPE software error in packetised TX.*

**#define IX\_HSSACC\_RX\_FRM\_SYNC\_ERR**  
*HSS port RX Frame Sync error.*

**#define IX\_HSSACC\_RX\_OVER\_RUN\_ERR**  
*HSS port RX over-run error.*

**#define IX\_HSSACC\_CHANNELISED\_SW\_RX\_ERR**  
*NPE software error in channelised RX.*

**#define IX\_HSSACC\_PACKETISED\_SW\_RX\_ERR**  
*NPE software error in packetised TX.*

**#define IX\_HSSACC\_PKT\_MIN\_RX\_MBUF\_SIZE**  
*Minimum size of the Rx mbuf in bytes which the client must supply to the component.*

## Typedefs

typedef

UINT32 **IxHssAccPktUserId**

*The client supplied value which will be supplied as a parameter with a given callback.*

typedef void(\* **IxHssAccLastErrorCallback** )(unsigned lastHssError, unsigned servicePort)  
*Prototype of the clients function to accept notification of the last error.*

typedef void(\* **IxHssAccPktRxCallback** )(IX\_MBUF \*buffer, unsigned numHssErrs, **IxHssAccPktStatus** pktStatus, **IxHssAccPktUserId** rxUserId)  
*Prototype of the clients function to accept notification of packetised rx.*

typedef void(\* **IxHssAccPktRxFreeLowCallback** )( **IxHssAccPktUserId** rxFreeLowUserId)  
*Prototype of the clients function to accept notification of requirement of more Rx Free buffers.*

typedef void(\* **IxHssAccPktTxDoneCallback** )(IX\_MBUF \*buffer, unsigned numHssErrs, **IxHssAccPktStatus** pktStatus, **IxHssAccPktUserId** txDoneUserId)  
*Prototype of the clients function to accept notification of completion with Tx buffers.*

typedef void(\* **IxHssAccChanRxCallback** )( **IxHssAccHssPort** hssPortId, unsigned rxOffset, unsigned txOffset, unsigned numHssErrs)  
*Prototype of the clients function to accept notification of channelised rx.*

## Enumerations

```
enum IxHssAccHssPort {  
    IX_HSSACC_HSS_PORT_0,  
    IX_HSSACC_HSS_PORT_1,  
    IX_HSSACC_HSS_PORT_MAX  
}
```

*The HSS port ID – There are two identical ports (0–1).*

```
enum IxHssAccHdlcPort {  
    IX_HSSACC_HDLC_PORT_0,  
    IX_HSSACC_HDLC_PORT_1,  
    IX_HSSACC_HDLC_PORT_2,  
    IX_HSSACC_HDLC_PORT_3,  
    IX_HSSACC_HDLC_PORT_MAX  
}
```

*The HDLC port ID – There are four identical HDLC ports (0–3) per HSS port and they correspond to the 4 E1/T1 trunks.*

```
enum IxHssAccTdmSlotUsage {  
    IX_HSSACC_TDMMAP_UNASSIGNED,  
    IX_HSSACC_TDMMAP_HDLC,  
    IX_HSSACC_TDMMAP_VOICE56K,  
    IX_HSSACC_TDMMAP_VOICE64K,  
    IX_HSSACC_TDMMAP_MAX  
}
```

*The HSS TDM stream timeslot assignment types.*

```
enum IxHssAccFrmSyncType {  
    IX_HSSACC_FRM_SYNC_ACTIVE_LOW,  
    IX_HSSACC_FRM_SYNC_ACTIVE_HIGH,  
    IX_HSSACC_FRM_SYNC_FALLINGEDGE,  
    IX_HSSACC_FRM_SYNC_RISINGEDGE,  
    IX_HSSACC_FRM_SYNC_TYPE_MAX  
}
```

*The HSS frame sync pulse type.*

```
enum IxHssAccFrmSyncEnable {  
    IX_HSSACC_FRM_SYNC_INPUT,  
    IX_HSSACC_FRM_SYNC_INVALID_VALUE,  
    IX_HSSACC_FRM_SYNC_OUTPUT_FALLING,  
    IX_HSSACC_FRM_SYNC_OUTPUT_RISING,  
    IX_HSSACC_FRM_SYNC_ENABLE_MAX  
}
```

*The IxHssAccFrmSyncEnable determines how the frame sync pulse is used.*

```
enum IxHssAccClkEdge {  
    IX_HSSACC_CLK_EDGE_FALLING,  
    IX_HSSACC_CLK_EDGE_RISING,  
    IX_HSSACC_CLK_EDGE_MAX  
}
```

*IxHssAccClkEdge is used to determine the clk edge to use for framing and data.*

```
enum IxHssAccClkDir {  
    IX_HSSACC_SYNC_CLK_DIR_INPUT,  
    IX_HSSACC_SYNC_CLK_DIR_OUTPUT,  
    IX_HSSACC_SYNC_CLK_DIR_MAX  
}
```

*The HSS clock direction.*

```
enum IxHssAccFrmPulseUsage {  
    IX_HSSACC_FRM_PULSE_ENABLED,  
    IX_HSSACC_FRM_PULSE_DISABLED,  
    IX_HSSACC_FRM_PULSE_MAX  
}
```

*The HSS frame pulse usage.*

```
enum IxHssAccDataRate {  
    IX_HSSACC_CLK_RATE,  
    IX_HSSACC_HALF_CLK_RATE,  
    IX_HSSACC_DATA_RATE_MAX  
}
```

*The HSS Data rate in relation to the clock.*

```
enum IxHssAccDataPolarity {  
    IX_HSSACC_DATA_POLARITY_SAME,  
    IX_HSSACC_DATA_POLARITY_INVERT,  
    IX_HSSACC_DATA_POLARITY_MAX  
}
```

*The HSS data polarity type.*

```
enum IxHssAccBitEndian {  
    IX_HSSACC_LSB_ENDIAN,  
    IX_HSSACC_MSB_ENDIAN,  
    IX_HSSACC_ENDIAN_MAX  
}
```

*HSS Data endianness.*

```
enum IxHssAccDrainMode {  
    IX_HSSACC_TX_PINS_NORMAL,  
    IX_HSSACC_TX_PINS_OPEN_DRAIN,  
    IX_HSSACC_TX_PINS_MAX  
}
```

*Tx pin open drain mode.*

```
enum IxHssAccSOFTType {  
    IX_HSSACC_SOF_FBIT,  
    IX_HSSACC_SOF_DATA,  
    IX_HSSACC_SOF_MAX  
}
```

*HSS start of frame types.*

```
enum IxHssAccDataEnable {
    IX_HSSACC_DE_TRI_STATE,
    IX_HSSACC_DE_DATA,
    IX_HSSACC_DE_MAX
}
```

*IxHssAccDataEnable is used to determine whether or not to drive the data pins.*

```
enum IxHssAccTxSigType {
    IX_HSSACC_TXSIG_LOW,
    IX_HSSACC_TXSIG_HIGH,
    IX_HSSACC_TXSIG_HIGH_IMP,
    IX_HSSACC_TXSIG_MAX
}
```

*IxHssAccTxSigType is used to determine how to drive the data pins.*

```
enum IxHssAccFbType {
    IX_HSSACC_FB_FIFO,
    IX_HSSACC_FB_HIGH_IMP,
    IX_HSSACC_FB_MAX
}
```

*IxHssAccFbType determines how to drive the Fbit.*

```
enum IxHssAcc56kEndianness {
    IX_HSSACC_56KE_BIT_7_UNUSED,
    IX_HSSACC_56KE_BIT_0_UNUSED,
    IX_HSSACC_56KE_MAX
}
```

*56k data endianness when using the 56k type*

```
enum IxHssAcc56kSel {
    IX_HSSACC_56KS_32_8_DATA,
    IX_HSSACC_56KS_56K_DATA,
    IX_HSSACC_56KS_MAX
}
```

*56k data transmission type when using the 56k type*

```
enum IxHssAccClkSpeed {
    IX_HSSACC_CLK_SPEED_512KHZ,
    IX_HSSACC_CLK_SPEED_1536KHZ,
    IX_HSSACC_CLK_SPEED_1544KHZ,
    IX_HSSACC_CLK_SPEED_1568KHZ,
    IX_HSSACC_CLK_SPEED_2048KHZ,
    IX_HSSACC_CLK_SPEED_4096KHZ,
    IX_HSSACC_CLK_SPEED_8192KHZ,
    IX_HSSACC_CLK_SPEED_MAX
}
```

*IxHssAccClkSpeed represents the HSS clock speeds available.*

```
enum IxHssAccPktStatus {
    IX_HSSACC_PKT_OK,
    IX_HSSACC_STOP_SHUTDOWN_ERROR,
}
```

```

    IX_HSSACC_HDLC_ALN_ERROR,
    IX_HSSACC_HDLC_FCS_ERROR,
    IX_HSSACC_RXFREE_Q_EMPTY_ERROR,
    IX_HSSACC_HDLC_MAX_FRAME_SIZE_EXCEEDED,
    IX_HSSACC_HDLC_ABORT_ERROR,
    IX_HSSACC_DISCONNECT_IN_PROGRESS
}

```

*Indicates the status of packets passed to the client.*

```

enum IxHssAccPktCrcType {
    IX_HSSACC_PKT_16_BIT_CRC,
    IX_HSSACC_PKT_32_BIT_CRC
}

```

*HDLC CRC type.*

```

enum IxHssAccPktHdlcIdleType {
    IX_HSSACC_HDLC_IDLE_ONES,
    IX_HSSACC_HDLC_IDLE_FLAGS
}

```

*HDLC idle transmission type.*

## Functions

**PUBLIC**  
**IX\_STATUS IxHssAccPortInit** (**IxHssAccHssPort** hssPortId, **IxHssAccConfigParams** \*configParams, **IX\_STATUS IxHssAccTdmSlotUsage** \*tdmMap, **IxHssAccLastErrorCallback** lastHssErrorCallback)  
*Initialise a HSS port. No channelised or packetised connections should exist in the HssAccess layer while this interface is being called.*

**PUBLIC**  
**IX\_STATUS ixHssAccLastErrorRetrievalInitiate** (**IxHssAccHssPort** hssPortId)  
*Initiate the retrieval of the last HSS error. The HSS port should be configured before attempting to call this interface.*

**PUBLIC**  
**IX\_STATUS ixHssAccInit** (void)  
*This function is responsible for initialising resources for use by the packetised and channelised clients. It should be called before any other HssAccess interface is called. No other HssAccPacketised interface should be called while this interface is being processed.*

**PUBLIC**  
**IX\_STATUS IxHssAccPktPortConnect** (**IxHssAccHssPort** hssPortId, **IxHssAccHdlcPort** hdlcPortId, **IX\_STATUS** BOOL hdlcFraming, **IxHssAccHdlcMode** hdlcMode, **IX\_STATUS** BOOL hdlcBitInvert, unsigned blockSizeInWords, **UINT32** rawIdleBlockPattern, **IxHssAccPktHdlcFraming** hdlcTxFraming, **IxHssAccPktHdlcFraming** hdlcRxFraming, unsigned frmFlagStart, **IxHssAccPktRxCallback** rxCallback, **IxHssAccPktUserId** rxUserId, **IxHssAccPktRxFreeLowCallback** rxFreeLowCallback, **IxHssAccPktUserId** rxFreeLowUserId, **IxHssAccPktTxDoneCallback** txDoneCallback, **IxHssAccPktUserId** txDoneUserId)  
*This function is responsible for connecting a client to one of the 4 available HDLC ports. The HSS port should be configured before attempting a connect. No other HssAccPacketised*

*interface should be called while this connect is being processed.*

**PUBLIC**

**IX\_STATUS ixHssAccPktPortEnable (IxHssAccHssPort hssPortId, IxHssAccHdlcPort hdlcPortId)**

*This function is responsible for enabling a packetised service for the specified HSS/HDLC port combination. It enables the RX flow. The client must have already connected to a packetised service and is responsible for ensuring an adequate amount of RX mbufs have been supplied to the access component before enabling the packetised service. This function must be called on a given port before any call to ixHssAccPktPortTx on the same port. No other HssAccPacketised interface should be called while this interface is being processed.*

**PUBLIC**

**IX\_STATUS ixHssAccPktPortDisable (IxHssAccHssPort hssPortId, IxHssAccHdlcPort hdlcPortId)**

*This function is responsible for disabling a packetised service for the specified HSS/HDLC port combination. It disables the RX flow. The client must have already connected to and enabled a packetised service for the specified HDLC port. This disable interface can be called before a disconnect, but is not required to.*

**PUBLIC**

**IX\_STATUS ixHssAccPktPortDisconnect (IxHssAccHssPort hssPortId, IxHssAccHdlcPort hdlcPortId)**

*This function is responsible for disconnecting a client from one of the 4 available HDLC ports. It is not required that the Rx Flow has been disabled before calling this function. If the RX Flow has not been disabled, the disconnect will disable it before proceeding with the disconnect. No other HssAccPacketised interface should be called while this interface is being processed.*

**PUBLIC ixHssAccPktPortIsDisconnectComplete (IxHssAccHssPort hssPortId, IxHssAccHdlcPort hdlcPortId)**  
**BOOL**

*This function is called to check if a given HSS/HDLC port combination is in a connected state or not. This function may be called at any time to determine a ports state. No other HssAccPacketised interface should be called while this interface is being processed.*

**PUBLIC ixHssAccPktPortRxFreeReplenish (IxHssAccHssPort hssPortId, IxHssAccHdlcPort hdlcPortId, IX\_STATUS hdlcPortId, IX\_MBUF \*buffer)**

*Function which the client calls at regular intervals to provide mbufs to the access component for RX. A connection should exist for the specified hssPortId/hdlcPortId combination before attempting to call this interface. Also, the connection should not be in a disconnecting state.*

**PUBLIC ixHssAccPktPortTx (IxHssAccHssPort hssPortId, IxHssAccHdlcPort hdlcPortId, IX\_STATUS IX\_MBUF \*buffer)**

*Function which the client calls when it wants to transmit packetised data. An enabled connection should exist on the specified hssPortId/hdlcPortId combination before attempting to call this interface. No other HssAccPacketised interface should be called while this interface is being processed.*

**PUBLIC ixHssAccChanConnect (IxHssAccHssPort hssPortId, unsigned bytesPerTSTrigger, UINT8 IX\_STATUS \*rxCircular, unsigned numRxBytesPerTS, UINT32 \*txPtrList, unsigned numTxPtrLists, unsigned numTxBytesPerBlk, IxHssAccChanRxCallback rxCallback)**

*This function allows the client to connect to the Tx/Rx NPE Channelised Service. There can only be one client per HSS port. The client is responsible for ensuring that the HSS port is configured appropriately before its connect request. No other HssAccChannelised interface*

*should be called while this interface is being processed.*

PUBLIC

IX\_STATUS **ixHssAccChanPortEnable** (**IxHssAccHssPort** hssPortId)

*This function is responsible for enabling a channelised service for the specified HSS port. It enables the NPE RX flow. The client must have already connected to a channelised service before enabling the channelised service. No other HssAccChannelised interface should be called while this interface is being processed.*

PUBLIC

IX\_STATUS **ixHssAccChanPortDisable** (**IxHssAccHssPort** hssPortId)

*This function is responsible for disabling a channelised service for the specified HSS port. It disables the NPE RX flow. The client must have already connected to and enabled a channelised service for the specified HSS port. This disable interface can be called before a disconnect, but is not required to. No other HssAccChannelised interface should be called while this interface is being processed.*

PUBLIC

IX\_STATUS **ixHssAccChanDisconnect** (**IxHssAccHssPort** hssPortId)

*This function allows the client to Disconnect from a channelised service. If the NPE RX Flow has not been disabled, the disconnect will disable it before proceeding with other disconnect functionality. No other HssAccChannelised interface should be called while this interface is being processed.*

PUBLIC **ixHssAccChanStatusQuery** (**IxHssAccHssPort** hssPortId, BOOL \*dataRecvd, unsigned

IX\_STATUS \*rxOffset, unsigned \*txOffset, unsigned \*numHssErrs)

*This function is called by the client to query whether or not channelised data has been received. If there is, hssChanAcc will return the details in the output parameters. An enabled connection should exist on the specified hssPortId before attempting to call this interface. No other HssAccChannelised interface should be called while this interface is being processed.*

PUBLIC void **ixHssAccShow** (void)

*This function will display the current state of the IxHssAcc component. The output is sent to stdout.*

PUBLIC void **ixHssAccStatsInit** (void)

*This function will reset the IxHssAcc statistics.*

---

## Detailed Description

The public API for the IXP425 HssAccess component.

IxHssAcc is the access layer to the HSS packetised and channelised services

### Design Notes

- The mbuf address is to be specified in the least significant 28-bit location [0:27]. The most significant 4-bit [31:29] are for internal use only.



- When a packet-pipe is configured for 56Kbps RAW mode, byte alignment of the transmitted data is not preserved. All raw data that is transmitted will be received in proper order by the receiver, but the first bit of the packet may be seen at any offset within a byte; all subsequent bytes will have the same offset for the duration of the packet. The same offset also applies to all subsequent packets received on the packet-pipe too. (Similar results will occur for data received from remote end.) While this behavior will also occur for 56Kbps HDLC mode, the HDLC encoding/decoding will preserve the original byte alignment at the receiver end.

### 56Kbps Packetised Service Bandwidth Limitation

- IxHssAcc supports 56Kbps packetised service at a maximum aggregate rate for all HSS ports/HDLC channels of 12.288Mbps[1] in each direction, i.e. it supports 56Kbps packetised service on up to 8 T1 trunks. It does not support 56Kbps packetised service on 8 E1 trunks (i.e. 4 trunks per HSS port) unless those trunks are running 'fractional E1' with maximum aggregate rate of 12.288 Mbps in each direction.

[1] 12.288Mbps = 1.536Mbp \* 8 T1

---

## Define Documentation

```
#define IX_HSSACC_CHANNELISED_SW_RX_ERR
```

NPE software error in channelised RX.

Definition at line **200** of file **IxHssAcc.h**.

```
#define IX_HSSACC_CHANNELISED_SW_TX_ERR
```

NPE software error in channelised TX.

Definition at line **172** of file **IxHssAcc.h**.

```
#define IX_HSSACC_NO_ERROR
```

HSS port no error present.

Definition at line **151** of file **IxHssAcc.h**.

```
#define IX_HSSACC_PACKETISED_SW_RX_ERR
```

NPE software error in packetised TX.

Definition at line **207** of file **IxHssAcc.h**.

```
#define IX_HSSACC_PACKETISED_SW_TX_ERR
```

NPE software error in packetised TX.

Definition at line **179** of file **IxHssAcc.h**.

```
#define IX_HSSACC_PARAM_ERR
```

HssAccess function return value for a parameter error.

Definition at line **118** of file **IxHssAcc.h**.

```
#define IX_HSSACC_PKT_DISCONNECTING
```

Indicates that a disconnect call is progressing and will disconnect soon.

Definition at line **133** of file **IxHssAcc.h**.

```
#define IX_HSSACC_PKT_MIN_RX_MBUF_SIZE
```

Minimum size of the Rx mbuf in bytes which the client must supply to the component.

Definition at line **219** of file **IxHssAcc.h**.

```
#define IX_HSSACC_Q_WRITE_OVERFLOW
```

Indicates that an attempt to Tx or to replenish an RxFree Q failed due to Q overflow.

Definition at line **141** of file **IxHssAcc.h**.

```
#define IX_HSSACC_RESOURCE_ERR
```

HssAccess function return value for a resource error.

Definition at line **125** of file **IxHssAcc.h**.

```
#define IX_HSSACC_RX_FRM_SYNC_ERR
```

HSS port RX Frame Sync error.

Definition at line **186** of file **IxHssAcc.h**.

```
#define IX_HSSACC_RX_OVER_RUN_ERR
```

HSS port RX over-run error.

Definition at line **193** of file **IxHssAcc.h**.

```
#define IX_HSSACC_TSLOTS_PER_HSS_PORT
```

The max number of TDM timeslots supported per HSS port –  $4E1's = 32 \times 4 = 128$ .

Definition at line **106** of file **IxHssAcc.h**.

```
#define IX_HSSACC_TX_FRM_SYNC_ERR
```

HSS port TX Frame Sync error.

Definition at line **158** of file **IxHssAcc.h**.

```
#define IX_HSSACC_TX_OVER_RUN_ERR
```

HSS port TX over-run error.

Definition at line **165** of file **IxHssAcc.h**.

---

## Typedef Documentation

```
IxHssAccChanRxCallback
```

Prototype of the clients function to accept notification of channelised rx.

This callback, if defined by the client in the connect, will get called in the context of an IRQ. The IRQ will be triggered when the hssSyncQMQ is not empty. The queued entry will be dequeued and this function will be executed.

### **Parameters:**

<i>IxHssAccHssPort</i>	hssPortId – The HSS port Id. There are two identical ports (0–1).
<i>unsigned</i>	txOffset (in) – an offset indicating from where within the txPtrList the NPE is currently transmitting from.
<i>unsigned</i>	rxOffset (in) – an offset indicating where within the receive buffers the NPE has just written the received data to.
<i>unsigned</i>	numHssErrs (in) – This is the number of hssErrors the Npe has received

### **Returns:**

void

Definition at line **737** of file **IxHssAcc.h**.

#### IxHssAccLastErrorCallback

Prototype of the clients function to accept notification of the last error.

This function is registered through the config. The client will initiate the last error retrieval. The HssAccess component will send a message to the NPE through the NPE Message Handler. When a response to the read is received, the NPE Message Handler will callback the HssAccess component which will execute this function in the same IxNpeMh context. The client will be passed the last error and the related service port (packetised 0–3, channelised 0)

**Parameters:**

*unsigned* lastHssError (in) – The last Hss error registered that has been registered.

*unsigned* servicePort (in) – This is the service port number. (packetised 0–3, channelised 0)

**Returns:**

void

Definition at line **646** of file **IxHssAcc.h**.

#### IxHssAccPktRxCallback

Prototype of the clients function to accept notification of packetised rx.

This function is registered through the ixHssAccPktPortConnect. hssPktAcc will pass received data in the form of mbufs to the client. The mbuf passed back to the client could contain a chain of buffers, depending on the packet size received.

**Parameters:**

*IX\_MBUF* \*buffer (in) – This is the mbuf which contains the payload received.

*unsigned* numHssErrors (in) – This is the number of hssErrors the Npe has received

*IxHssAccPktStatus* pktStatus (in) – This is the status of the mbuf that has been received.

*IxHssAccPktUserId* rxUserId (in) – This is the client supplied value passed in at ixHssAccPktPortConnect time which is now returned to the client.

**Returns:**

void

Definition at line **670** of file **IxHssAcc.h**.

#### IxHssAccPktRxFreeLowCallback

Prototype of the clients function to accept notification of requirement of more Rx Free buffers.

The client can choose to register a callback of this type when calling a connecting. This function is

registered through the `ixHssAccPktPortConnect`. If defined, the access layer will provide the trigger for this callback. The callback will be responsible for supplying mbufs to the access layer for use on the receive path from the HSS using `ixHssPktAccFreeBufReplenish`.

**Returns:**

void

Definition at line **689** of file **IxHssAcc.h**.

## IxHssAccPktTxDoneCallback

Prototype of the clients function to accept notification of completion with Tx buffers.

This function is registered through the `ixHssAccPktPortConnect`. It enables the `hssPktAcc` to pass buffers back to the client when transmission is complete.

**Parameters:**

<i>IX_MBUF</i>	<code>*buffer</code> (in) – This is the mbuf which contained the payload that was for Tx.
<i>unsigned</i>	<code>numHssErrs</code> (in) – This is the number of <code>hssErrors</code> the Npe has received
<i>IxHssAccPktStatus</i>	<code>pktStatus</code> (in) – This is the status of the mbuf that has been transmitted.
<i>IxHssAccPktUserId</i>	<code>txDoneUserId</code> (in) – This is the client supplied value passed in at <code>ixHssAccPktPortConnect</code> time which is now returned to the client.

**Returns:**

void

Definition at line **711** of file **IxHssAcc.h**.

## UINT32 IxHssAccPktUserId

The client supplied value which will be supplied as a parameter with a given callback.

This value will be passed into the `ixHssAccPktPortConnect` function once each with given callbacks. This value will then be passed back to the client as one of the parameters to each of these callbacks, when these callbacks are called.

Definition at line **623** of file **IxHssAcc.h**.

---

# Enumeration Type Documentation

## enum IxHssAcc56kEndianness

56k data endianness when using the 56k type

**Enumeration values:**

*IX\_HSSACC\_56KE\_BIT\_7\_UNUSED*

	High bit is unused.
<i>IX_HSSACC_56KE_BIT_0_UNUSED</i>	Low bit is unused.
<i>IX_HSSACC_56KE_MAX</i>	Delimiter for error checks.

Definition at line **441** of file **IxHssAcc.h**.

```
enum IxHssAcc56kSel
```

56k data transmission type when using the 56k type

***Enumeration values:***

<i>IX_HSSACC_56KS_32_8_DATA</i>	32/8 bit data
<i>IX_HSSACC_56KS_56K_DATA</i>	56K data
<i>IX_HSSACC_56KS_MAX</i>	Delimiter for error checks.

Definition at line **453** of file **IxHssAcc.h**.

```
enum IxHssAccBitEndian
```

HSS Data endianness.

***Enumeration values:***

<i>IX_HSSACC_LSB_ENDIAN</i>	TX/RX Least Significant Bit first.
<i>IX_HSSACC_MSB_ENDIAN</i>	TX/RX Most Significant Bit first.
<i>IX_HSSACC_ENDIAN_MAX</i>	Delimiter for the purposes of error checks.

Definition at line **364** of file **IxHssAcc.h**.

```
enum IxHssAccClkDir
```

The HSS clock direction.

***Enumeration values:***

<i>IX_HSSACC_SYNC_CLK_DIR_INPUT</i>	Clock is an input.
<i>IX_HSSACC_SYNC_CLK_DIR_OUTPUT</i>	Clock is an output.
<i>IX_HSSACC_SYNC_CLK_DIR_MAX</i>	Delimiter for error checks.

Definition at line **314** of file **IxHssAcc.h**.

## enum IxHssAccClkEdge

IxHssAccClkEdge is used to determine the clk edge to use for framing and data.

### **Enumeration values:**

<i>IX_HSSACC_CLK_EDGE_FALLING</i>	Clock sampled off a falling edge.
<i>IX_HSSACC_CLK_EDGE_RISING</i>	Clock sampled off a rising edge.
<i>IX_HSSACC_CLK_EDGE_MAX</i>	Delimiter for error checks.

Definition at line **302** of file **IxHssAcc.h**.

## enum IxHssAccClkSpeed

IxHssAccClkSpeed represents the HSS clock speeds available.

### **Enumeration values:**

<i>IX_HSSACC_CLK_SPEED_512KHZ</i>	512KHz
<i>IX_HSSACC_CLK_SPEED_1536KHZ</i>	1.536MHz
<i>IX_HSSACC_CLK_SPEED_1544KHZ</i>	1.544MHz
<i>IX_HSSACC_CLK_SPEED_1568KHZ</i>	1.568MHz
<i>IX_HSSACC_CLK_SPEED_2048KHZ</i>	2.048MHz
<i>IX_HSSACC_CLK_SPEED_4096KHZ</i>	4.096MHz
<i>IX_HSSACC_CLK_SPEED_8192KHZ</i>	8.192MHz
<i>IX_HSSACC_CLK_SPEED_MAX</i>	Delimiter for error checking.

Definition at line **466** of file **IxHssAcc.h**.

## enum IxHssAccDataEnable

IxHssAccDataEnable is used to determine whether or not to drive the data pins.

### **Enumeration values:**

<i>IX_HSSACC_DE_TRI_STATE</i>	TRI-State the data pins.
<i>IX_HSSACC_DE_DATA</i>	Push data out the data pins.
<i>IX_HSSACC_DE_MAX</i>	Delimiter for error checks.

Definition at line **402** of file **IxHssAcc.h**.

## enum IxHssAccDataPolarity

The HSS data polarity type.

### **Enumeration values:**

<i>IX_HSSACC_DATA_POLARITY_SAME</i>	Don't invert data between NPE and HSS
-------------------------------------	---------------------------------------

	FIFOs.
<i>IX_HSSACC_DATA_POLARITY_INVERT</i>	Invert data between NPE and HSS FIFOs.
<i>IX_HSSACC_DATA_POLARITY_MAX</i>	Delimiter for error checks.

Definition at line **350** of file **IxHssAcc.h**.

```
enum IxHssAccDataRate
```

The HSS Data rate in relation to the clock.

**Enumeration values:**

<i>IX_HSSACC_CLK_RATE</i>	Data rate is at the configured clk speed.
<i>IX_HSSACC_HALF_CLK_RATE</i>	Data rate is half the configured clk speed.
<i>IX_HSSACC_DATA_RATE_MAX</i>	Delimiter for error checks.

Definition at line **338** of file **IxHssAcc.h**.

```
enum IxHssAccDrainMode
```

Tx pin open drain mode.

**Enumeration values:**

<i>IX_HSSACC_TX_PINS_NORMAL</i>	Normal mode.
<i>IX_HSSACC_TX_PINS_OPEN_DRAIN</i>	Open Drain mode.
<i>IX_HSSACC_TX_PINS_MAX</i>	Delimiter for error checks.

Definition at line **377** of file **IxHssAcc.h**.

```
enum IxHssAccFbType
```

IxHssAccFbType determines how to drive the Fbit.

**Warning:**

This will only be used for T1 @ 1.544MHz

**Enumeration values:**

<i>IX_HSSACC_FB_FIFO</i>	Fbit is dictated in FIFO.
<i>IX_HSSACC_FB_HIGH_IMP</i>	



<i>IX_HSSACC_FB_MAX</i>	Fbit is high impedance. Delimiter for error checks.
-------------------------	--

Definition at line **429** of file **IxHssAcc.h**.

```
enum IxHssAccFrmPulseUsage
```

The HSS frame pulse usage.

**Enumeration values:**

<i>IX_HSSACC_FRM_PULSE_ENABLED</i>	Generate/Receive frame pulses.
<i>IX_HSSACC_FRM_PULSE_DISABLED</i>	Disregard frame pulses.
<i>IX_HSSACC_FRM_PULSE_MAX</i>	Delimiter for error checks.

Definition at line **326** of file **IxHssAcc.h**.

```
enum IxHssAccFrmSyncEnable
```

The IxHssAccFrmSyncEnable determines how the frame sync pulse is used.

**Enumeration values:**

<i>IX_HSSACC_FRM_SYNC_INPUT</i>	Frame sync is sampled as an input.
<i>IX_HSSACC_FRM_SYNC_INVALID_VALUE</i>	1 is not used
<i>IX_HSSACC_FRM_SYNC_OUTPUT_FALLING</i>	Frame sync is an output generated off a falling clock edge.
<i>IX_HSSACC_FRM_SYNC_OUTPUT_RISING</i>	Frame sync is an output generated off a rising clock edge.
<i>IX_HSSACC_FRM_SYNC_ENABLE_MAX</i>	Delimiter for error checks.

Definition at line **285** of file **IxHssAcc.h**.

```
enum IxHssAccFrmSyncType
```

The HSS frame sync pulse type.

**Enumeration values:**

<i>IX_HSSACC_FRM_SYNC_ACTIVE_LOW</i>	Frame sync is sampled low.
<i>IX_HSSACC_FRM_SYNC_ACTIVE_HIGH</i>	sampled high
<i>IX_HSSACC_FRM_SYNC_FALLINGEDGE</i>	sampled on a falling edge
<i>IX_HSSACC_FRM_SYNC_RISINGEDGE</i>	

<i>IX_HSSACC_FRM_SYNC_TYPE_MAX</i>	sampled on a rising edge Delimiter for error checks.
------------------------------------	---

Definition at line **271** of file **IxHssAcc.h**.

#### enum IxHssAccHdlcPort

The HDLC port ID – There are four identical HDLC ports (0–3) per HSS port and they correspond to the 4 E1/T1 trunks.

##### *Enumeration values:*

<i>IX_HSSACC_HDLC_PORT_0</i>	HDLC Port 0.
<i>IX_HSSACC_HDLC_PORT_1</i>	HDLC Port 1.
<i>IX_HSSACC_HDLC_PORT_2</i>	HDLC Port 2.
<i>IX_HSSACC_HDLC_PORT_3</i>	HDLC Port 3.
<i>IX_HSSACC_HDLC_PORT_MAX</i>	Delimiter for error checks.

Definition at line **243** of file **IxHssAcc.h**.

#### enum IxHssAccHssPort

The HSS port ID – There are two identical ports (0–1).

##### *Enumeration values:*

<i>IX_HSSACC_HSS_PORT_0</i>	HSS Port 0.
<i>IX_HSSACC_HSS_PORT_1</i>	HSS Port 1.
<i>IX_HSSACC_HSS_PORT_MAX</i>	Delimiter for error checks.

Definition at line **230** of file **IxHssAcc.h**.

#### enum IxHssAccPktCrcType

HDLC CRC type.

##### *Enumeration values:*

<i>IX_HSSACC_PKT_16_BIT_CRC</i>	16 bit CRC is being used
<i>IX_HSSACC_PKT_32_BIT_CRC</i>	32 bit CRC is being used

Definition at line **507** of file **IxHssAcc.h**.

## enum IxHssAccPktHdlcIdleType

HDLC idle transmission type.

### **Enumeration values:**

<i>IX_HSSACC_HDLC_IDLE_ONES</i>	idle tx/rx will be a succession of ones
<i>IX_HSSACC_HDLC_IDLE_FLAGS</i>	idle tx/rx will be repeated flags

Definition at line **518** of file **IxHssAcc.h**.

## enum IxHssAccPktStatus

Indicates the status of packets passed to the client.

### **Enumeration values:**

<i>IX_HSSACC_PKT_OK</i>	Error free.
<i>IX_HSSACC_STOP_SHUTDOWN_ERROR</i>	Errored due to stop or shutdown occurrence.
<i>IX_HSSACC_HDLC_ALN_ERROR</i>	HDLC alignment error.
<i>IX_HSSACC_HDLC_FCS_ERROR</i>	HDLC Frame Check Sum error.
<i>IX_HSSACC_RXFREE_Q_EMPTY_ERROR</i>	RxFree Q became empty while receiving this packet.
<i>IX_HSSACC_HDLC_MAX_FRAME_SIZE_EXCEEDED</i>	HDLC frame size received is greater than max specified at connect.
<i>IX_HSSACC_HDLC_ABORT_ERROR</i>	HDLC frame received is invalid due to an abort sequence received.
<i>IX_HSSACC_DISCONNECT_IN_PROGRESS</i>	Packet returned because a disconnect is in progress.

Definition at line **483** of file **IxHssAcc.h**.

## enum IxHssAccSOFTType

HSS start of frame types.

### **Enumeration values:**

<i>IX_HSSACC_SOF_FBIT</i>	Framing bit transmitted and expected on rx.
<i>IX_HSSACC_SOF_DATA</i>	Framing bit not transmitted nor expected on rx.
<i>IX_HSSACC_SOF_MAX</i>	Delimiter for error checks.

Definition at line **389** of file **IxHssAcc.h**.

```
enum IxHssAccTdmSlotUsage
```

The HSS TDM stream timeslot assignment types.

**Enumeration values:**

<i>IX_HSSACC_TDMMAP_UNASSIGNED</i>	Unassigned.
<i>IX_HSSACC_TDMMAP_HDLC</i>	HDLC – packetised.
<i>IX_HSSACC_TDMMAP_VOICE56K</i>	Voice56K – channelised.
<i>IX_HSSACC_TDMMAP_VOICE64K</i>	Voice64K – channelised.
<i>IX_HSSACC_TDMMAP_MAX</i>	Delimiter for error checks.

Definition at line **257** of file **IxHssAcc.h**.

```
enum IxHssAccTxSigType
```

IxHssAccTxSigType is used to determine how to drive the data pins.

**Enumeration values:**

<i>IX_HSSACC_TXSIG_LOW</i>	Drive the data pins low.
<i>IX_HSSACC_TXSIG_HIGH</i>	Drive the data pins high.
<i>IX_HSSACC_TXSIG_HIGH_IMP</i>	Drive the data pins with high impedance.
<i>IX_HSSACC_TXSIG_MAX</i>	Delimiter for error checks.

Definition at line **414** of file **IxHssAcc.h**.

---

## Function Documentation

```
IX_STATUS ixHssAccChanConnect ( IxHssAccHssPort      hssPortId,  
                                unsigned                bytesPerTSTrigger,  
                                UINT8 *                 rxCircular,  
                                unsigned                numRxBytesPerTS,  
                                UINT32 *                txPtrList,  
                                unsigned                numTxPtrLists,  
                                unsigned                numTxBytesPerBlk,  
                                IxHssAccChanRxCallback rxCallback  
                                )
```

This function allows the client to connect to the Tx/Rx NPE Channelised Service. There can only be one client per HSS port. The client is responsible for ensuring that the HSS port is configured appropriately before its connect request. No other HssAccChannelised interface should be called while this interface is being processed.

**Parameters:**

<i>IxHssAccHssPort</i>	hssPortId (in) – The HSS port Id. There are two identical ports (0–1).
<i>unsigned</i>	bytesPerTSTrigger (in) – The NPE will trigger the access component after bytesPerTSTrigger have been received for all trunk timeslots. This figure is a multiple of 8 e.g. 8 for 1ms trigger, 16 for 2ms trigger.
<i>UINT8</i>	*rxCircular (in) – A pointer to memory allocated by the client to be filled by data received. The buffer at this address is part of a pool of buffers to be accessed in a circular fashion. This address will be written to by the NPE. Therefore, it needs to be a physical address.
<i>unsigned</i>	numRxBytesPerTS (in) – The number of bytes allocated per timeslot within the receive memory. This figure will depend on the latency of the system. It needs to be deep enough for data to be read by the client before the NPE re–writes over that memory e.g. if the client samples at a rate of 40bytes per timeslot, numRxBytesPerTS may need to be 40bytes * 3. This would give the client 3 * 5ms of time before received data is over–written.
<i>UINT32</i>	*txPtrList (in) – The address of an area of contiguous memory allocated by the client to be populated with pointers to data for transmission. Each pointer list contains a pointer per active channel. The txPtrs will point to data to be transmitted by the NPE. Therefore, they must point to physical addresses.
<i>unsigned</i>	numTxPtrLists (in) – The number of pointer lists in txPtrList. This figure is dependent on jitter.
<i>unsigned</i>	numTxBytesPerBlk (in) – The size of the Tx data, in bytes, that each pointer within the PtrList points to.
<i>IxHssAccChanRxCallback</i>	rxCallback (in) – A client function pointer to be called back to handle the actual tx/rx of channelised data. If this is not NULL, an ISR will call this function. If this pointer is NULL, it implies that the client will use a polling mechanism to detect when the tx and rx of channelised data is to occur. The client will use hssChanAccStatus for this.

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error

**IX\_STATUS ixHssAccChanDisconnect ( **IxHssAccHssPort** hssPortId )**

This function allows the client to Disconnect from a channelised service. If the NPE RX Flow has not been disabled, the disconnect will disable it before proceeding with other disconnect functionality. No other HssAccChannelised interface should be called while this interface is being processed.

**Parameters:**

*IxHssAccHssPort* hssPortId(in) – The HSS port Id. There are two identical ports (0–1).

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error

IX\_STATUS ixHssAccChanPortDisable ( **IxHssAccHssPort** hssPortId )

This function is responsible for disabling a channelised service for the specified HSS port. It disables the NPE RX flow. The client must have already connected to and enabled a channelised service for the specified HSS port. This disable interface can be called before a disconnect, but is not required to. No other HssAccChannelised interface should be called while this interface is being processed.

**Parameters:**

*IxHssAccHssPort* hssPortId (in) – The HSS port Id. There are two identical ports (0–1).

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error

IX\_STATUS ixHssAccChanPortEnable ( **IxHssAccHssPort** hssPortId )

This function is responsible for enabling a channelised service for the specified HSS port. It enables the NPE RX flow. The client must have already connected to a channelised service before enabling the channelised service. No other HssAccChannelised interface should be called while this interface is being processed.

**Parameters:**

*IxHssAccHssPort* hssPortId (in) – The HSS port Id. There are two identical ports (0–1).

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error

IX\_STATUS ixHssAccChanStatusQuery ( **IxHssAccHssPort** hssPortId,  
  BOOL \*              dataRecvd,  
                                  unsigned \*              rxOffset,  
                                  unsigned \*              txOffset,  
                                  unsigned \*              numHssErrs

)

This function is called by the client to query whether or not channelised data has been received. If there is, hssChanAcc will return the details in the output parameters. An enabled connection should exist on the specified hssPortId before attempting to call this interface. No other HssAccChannelised interface should be called while this interface is being processed.

**Parameters:**

<i>IxHssAccHssPort</i>	hssPortId (in) – The HSS port Id. There are two identical ports (0–1).
<i>BOOL</i>	*dataRecvd (out) – This BOOL indicates to the client whether or not the access component has read any data for the client. If FALSE, the other output parameters will not have been written to.
<i>unsigned</i>	*rxOffset (out) – An offset to indicate to the client where within the receive buffers the NPE has just written the received data to.
<i>unsigned</i>	*txOffset (out) – An offset to indicate to the client from where within the txPtrList the NPE is currently transmitting from
<i>unsigned</i>	*numHssErrs (out) – The total number of HSS port errors since initial port configuration

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error

IX\_STATUS ixHssAccInit ( void )

This function is responsible for initialising resources for use by the packetised and channelised clients. It should be called before any other HssAccess interface is called. No other HssAccPacketised interface should be called while this interface is being processed.

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_RESOURCE\_ERR The function did not execute successfully due to a resource error

IX\_STATUS ixHssAccLastErrorRetrievalInitiate ( **IxHssAccHssPort** hssPortId )

Initiate the retrieval of the last HSS error. The HSS port should be configured before attempting to call this interface.

**Parameters:**

*IxHssAccHssPort* hssPortId (in) – the HSS port ID

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully

- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error

```

ixHssAccPktPortConnect ( IxHssAccHssPort          hssPortId,
                        IxHssAccHdlcPort          hdlcPortId,
                        BOOL                      hdlcFraming,
                        IxHssAccHdlcMode          hdlcMode,
                        BOOL                      hdlcBitInvert,
                        unsigned                  blockSizeInWords,
                        UINT32                  rawIdleBlockPattern,
                        IxHssAccPktHdlcFraming      hdlcTxFraming,
                        IxHssAccPktHdlcFraming      hdlcRxFraming,
                        unsigned                  frmFlagStart,
                        IxHssAccPktRxCallback        rxCallback,
                        IxHssAccPktUserId          rxUserId,
                        IxHssAccPktRxFreeLowCallback rxFreeLowCallback,
                        IxHssAccPktUserId          rxFreeLowUserId,
                        IxHssAccPktTxDoneCallback    txDoneCallback,
                        IxHssAccPktUserId          txDoneUserId
                        )

```

This function is responsible for connecting a client to one of the 4 available HDLC ports. The HSS port should be configured before attempting a connect. No other HssAccPacketised interface should be called while this connect is being processed.

**Parameters:**

<i>IxHssAccHssPort</i>	hssPortId (in) – The HSS port Id. There are two identical ports (0–1).
<i>IxHssAccHdlcPort</i>	hdlcPortId (in) – This is the number of the HDLC port and it corresponds to the physical E1/T1 trunk i.e. 0, 1, 2, 3
<i>BOOL</i>	hdlcFraming (in) – This value determines whether the service will use HDLC data or the debug, raw data type i.e. no HDLC processing
<i>IxHssAccHdlcMode</i>	hdlcMode (in) – This structure contains 56Kbps, HDLC–mode configuration parameters
<i>BOOL</i>	hdlcBitInvert (in) – This value determines whether bit inversion will occur between HDLC and HSS co–processors i.e. post–HDLC processing for transmit and pre–HDLC processing for receive, for the specified HDLC Termination Point
<i>unsigned</i>	blockSizeInWords (in) – The max tx/rx block size
<i>UINT32</i>	rawIdleBlockPattern (in) – Tx idle pattern in raw mode
<i>IxHssAccHdlcFraming</i>	hdlcTxFraming (in) – This structure contains the following information required by the NPE to configure the HDLC co–processor for TX
<i>IxHssAccHdlcFraming</i>	hdlcRxFraming (in) – This structure contains the following information required by the NPE to configure the HDLC co–processor for RX



<i>unsigned</i>	frmFlagStart – Number of flags to precede to transmitted flags (0–2).
<i>IxHssAccPktRxCallback</i>	rxCallback (in) – Pointer to the clients packet receive function.
<i>IxHssAccPktUserId</i>	rxUserId (in) – The client supplied rx value to be passed back as an argument to the supplied rxCallback
<i>IxHssAccPktRxFreeLowCallback</i>	rxFreeLowCallback (in) – Pointer to the clients Rx free buffer request function. If NULL, assume client will trigger independently.
<i>IxHssAccPktUserId</i>	rxFreeLowUserId (in) – The client supplied RxFreeLow value to be passed back as an argument to the supplied rxFreeLowCallback
<i>IxHssAccPktTxDoneCallback</i>	txDoneCallback (in) – Pointer to the clients Tx done callback function
<i>IxHssAccPktUserId</i>	txDoneUserId (in) – The client supplied txDone value to be passed back as an argument to the supplied txDoneCallback

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error
- ◇ IX\_HSSACC\_RESOURCE\_ERR The function did not execute successfully due to a resource error

```
IX_STATUS ixHssAccPktPortDisable ( IxHssAccHssPort  hssPortId,
                                   IxHssAccHdlcPort hdlcPortId
                                   )
```

This function is responsible for disabling a packetised service for the specified HSS/HDLC port combination. It disables the RX flow. The client must have already connected to and enabled a packetised service for the specified HDLC port. This disable interface can be called before a disconnect, but is not required to.

**Parameters:**

- IxHssAccHssPort* hssPortId (in) – The HSS port Id. There are two identical ports (0–1).
- IxHssAccHdlcPort* hdlcPortId (in) – The port id (0,1,2,3) to disable the service on.

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error

```
IX_STATUS ixHssAccPktPortDisconnect ( IxHssAccHssPort  hssPortId,
                                       IxHssAccHdlcPort hdlcPortId
                                       )
```

This function is responsible for disconnecting a client from one of the 4 available HDLC ports. It is not required that the Rx Flow has been disabled before calling this function. If the RX Flow has not been disabled, the disconnect will disable it before proceeding with the disconnect. No other HssAccPacketised interface should be called while this interface is being processed.

**Parameters:**

*IxHssAccHssPort* hssPortId (in) – The HSS port Id. There are two identical ports (0–1).

*IxHssAccHdlcPort* hdlcPortId (in) – This is the number of the HDLC port to disconnect and it corresponds to the physical E1/T1 trunk i.e. 0, 1, 2, 3

**Returns:**

◇ IX\_SUCCESS The function executed successfully

◇ IX\_FAIL The function did not execute successfully

◇ IX\_HSSACC\_PKT\_DISCONNECTING The function has initiated the disconnecting procedure but it has not completed yet.

```
IX_STATUS ixHssAccPktPortEnable ( IxHssAccHssPort  hssPortId,
                                   IxHssAccHdlcPort hdlcPortId
                                   )
```

This function is responsible for enabling a packetised service for the specified HSS/HDLC port combination. It enables the RX flow. The client must have already connected to a packetised service and is responsible for ensuring an adequate amount of RX mbufs have been supplied to the access component before enabling the packetised service. This function must be called on a given port before any call to ixHssAccPktPortTx on the same port. No other HssAccPacketised interface should be called while this interface is being processed.

**Parameters:**

*IxHssAccHssPort* hssPortId (in) – The HSS port Id. There are two identical ports (0–1).

*IxHssAccHdlcPort* hdlcPortId (in) – The port id (0,1,2,3) to enable the service on.

**Returns:**

◇ IX\_SUCCESS The function executed successfully

◇ IX\_FAIL The function did not execute successfully

◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error

```
BOOL ixHssAccPktPortIsDisconnectComplete ( IxHssAccHssPort  hssPortId,
                                             IxHssAccHdlcPort hdlcPortId
                                             )
```

This function is called to check if a given HSS/HDLC port combination is in a connected state or not. This function may be called at any time to determine a ports state. No other HssAccPacketised interface should be called while this interface is being processed.

**Parameters:**

*IxHssAccHssPort* hssPortId (in) – The HSS port Id. There are two identical ports (0–1).

*IxHssAccHdlcPort*

hdlcPortId (in) – This is the number of the HDLC port to disconnect and it corresponds to the physical E1/T1 trunk i.e. 0, 1, 2, 3

**Returns:**

- ◇ TRUE The state of this HSS/HDLC port combination is disconnected, so if a disconnect was called, it is now completed.
- ◇ FALSE The state of this HSS/HDLC port combination is connected, so if a disconnect was called, it is not yet completed.

```
IX_STATUS ixHssAccPktPortRxFreeReplenish ( IxHssAccHssPort  hssPortId,  
                                           IxHssAccHdlcPort hdlcPortId,  
                                           IX_MBUF *      buffer  
                                           )
```

Function which the client calls at regular intervals to provide mbufs to the access component for RX. A connection should exist for the specified hssPortId/hdlcPortId combination before attempting to call this interface. Also, the connection should not be in a disconnecting state.

**Parameters:**

<i>IxHssAccHssPort</i>	hssPortId (in) – The HSS port Id. There are two identical ports (0–1).
<i>IxHssAccHdlcPort</i>	hdlcPortId (in) – This is the number of the HDLC port and it corresponds to the physical E1/T1 trunk i.e. 0, 1, 2, 3
<i>IX_MBUF</i>	*buffer (in) – A pointer to a free mbuf to filled with payload.

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error
- ◇ IX\_HSSACC\_RESOURCE\_ERR The function did not execute successfully due to a resource error
- ◇ IX\_HSSACC\_Q\_WRITE\_OVERFLOW The function did not succeed due to a Q overflow

```
IX_STATUS ixHssAccPktPortTx ( IxHssAccHssPort  hssPortId,  
                             IxHssAccHdlcPort hdlcPortId,  
                             IX_MBUF *      buffer  
                             )
```

Function which the client calls when it wants to transmit packetised data. An enabled connection should exist on the specified hssPortId/hdlcPortId combination before attempting to call this interface. No other HssAccPacketised interface should be called while this interface is being processed.

**Parameters:**

<i>IxHssAccHssPort</i>	hssPortId (in) – The HSS port Id. There are two identical ports (0–1).
<i>IxHssAccHdlcPort</i>	hdlcPortId (in) – This is the number of the HDLC port and it corresponds to the physical E1/T1 trunk i.e. 0, 1, 2, 3
<i>IX_MBUF</i>	*buffer (in) – A pointer to a chain of mbufs which the client has filled with the payload

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error
- ◇ IX\_HSSACC\_RESOURCE\_ERR The function did not execute successfully due to a resource error. See note.
- ◇ IX\_HSSACC\_Q\_WRITE\_OVERFLOW The function did not succeed due to a Q overflow

**Note:**

IX\_HSSACC\_RESOURCE\_ERR is returned when a free descriptor cannot be obtained to send the chain of mbufs to the NPE. This is a normal scenario. HssAcc has a pool of descriptors and this error means that they are currently all in use. The recommended approach to this is to retry until a descriptor becomes free and the packet is successfully transmitted. Alternatively, the user could wait until the next IxHssAccPktTxDoneCallback callback is triggered, and then retry, as it is this event that causes a transmit descriptor to be freed.

```
IX_STATUS ixHssAccPortInit ( IxHssAccHssPort          hssPortId,
                           IxHssAccConfigParams *    configParams,
                           IxHssAccTdmSlotUsage *    tdmMap,
                           IxHssAccLastErrorCallback lastHssErrorCallback
                           )
```

Initialise a HSS port. No channelised or packetised connections should exist in the HssAccess layer while this interface is being called.

**Parameters:**

<i>IxHssAccHssPort</i>	<i>hssPortId</i> (in) – The HSS port Id. There are two identical ports (0–1).
<i>IxHssAccConfigParams</i>	<i>*configParams</i> (in) – A pointer to the HSS configuration structure
<i>IxHssAccTdmSlotUsage</i>	<i>*tdmMap</i> (in) – A pointer to an array of size IX_HSSACC_TSLOTS_PER_HSS_PORT, defining the slot usage over the HSS port
<i>IxHssAccLastErrorCallback</i>	<i>lastHssErrorCallback</i> (in) – Client callback to report last error

**Returns:**

- ◇ IX\_SUCCESS The function executed successfully
- ◇ IX\_FAIL The function did not execute successfully
- ◇ IX\_HSSACC\_PARAM\_ERR The function did not execute successfully due to a parameter error

```
void ixHssAccShow ( void )
```

This function will display the current state of the IxHssAcc component. The output is sent to stdout.

**Returns:**

void

```
void ixHssAccStatsInit ( void )
```

This function will reset the IxHssAcc statistics.

***Returns:***

void

# IXP425 NPE–A (IxNpeA) API

The Public API for the IXP425 NPE–A.

## Data Structures

struct **IxNpeA\_NpePacketDescriptor**  
*HSS Packetized NpePacket Descriptor Structure.*

struct **IxNpeA\_RxAtmVc**  
*Rx Descriptor definition.*

struct **IxNpeA\_TxAtmVc**  
*Tx Descriptor definition.*

## Defines

#define **IX\_NPE\_A\_MSSG\_ATM\_UTOPIA\_CONFIG\_WRITE**  
*ATM Message ID command to write the data to the offset in the Utopia Configuration Table.*

#define **IX\_NPE\_A\_MSSG\_ATM\_UTOPIA\_CONFIG\_LOAD**  
*ATM Message ID command triggers the NPE to copy the Utopia Configuration Table to the Utopia coprocessor.*

#define **IX\_NPE\_A\_MSSG\_ATM\_UTOPIA\_STATUS\_UPLOAD**  
*ATM Message ID command triggers the NPE to read–back the Utopia status registers and update the Utopia Status Table.*

#define **IX\_NPE\_A\_MSSG\_ATM\_UTOPIA\_STATUS\_READ**  
*ATM Message ID command to read the Utopia Status Table at the specified offset.*

#define **IX\_NPE\_A\_MSSG\_ATM\_TX\_ENABLE**  
*ATM Message ID command triggers the NPE to re–enable processing of any entries on the TxVcQ for this port.*

#define **IX\_NPE\_A\_MSSG\_ATM\_TX\_DISABLE**  
*ATM Message ID command triggers the NPE to disable processing on this port.*

#define **IX\_NPE\_A\_MSSG\_ATM\_RX\_ENABLE**  
*ATM Message ID command triggers the NPE to process any received cells for this VC according to the VC Lookup Table.*

#define **IX\_NPE\_A\_MSSG\_ATM\_RX\_DISABLE**  
*ATM Message ID command triggers the NPE to disable processing for this VC.*

#define **IX\_NPE\_A\_MSSG\_ATM\_STATUS\_READ**

*ATM Message ID command to read the ATM status. The data is returned via a response message.*

```
#define IX_NPE_A_MSSG_HSS_PORT_CONFIG_WRITE  
    HSS Message ID command writes the ConfigWord value to the location in the  
    HSS_CONFIG_TABLE specified by offset for HSS port hPort.  
  
#define IX_NPE_A_MSSG_HSS_PORT_CONFIG_LOAD  
    HSS Message ID command triggers the NPE to copy the contents of the HSS Configuration Table to  
    the appropriate configuration registers in the HSS coprocessor for the port specified by hPort.  
  
#define IX_NPE_A_MSSG_HSS_PORT_ERROR_READ  
    HSS Message ID command triggers the NPE to return an HssErrorReadResponse message for HSS  
    port hPort.  
  
#define IX_NPE_A_MSSG_HSS_CHAN_FLOW_ENABLE  
    HSS Message ID command triggers the NPE to reset internal status and enable the HssChannelized  
    operation on the HSS port specified by hPort.  
  
#define IX_NPE_A_MSSG_HSS_CHAN_FLOW_DISABLE  
    HSS Message ID command triggers the NPE to disable the HssChannelized operation on the HSS  
    port specified by hPort.  
  
#define IX_NPE_A_MSSG_HSS_CHAN_IDLE_PATTERN_WRITE  
    HSS Message ID command writes the HSSnC_IDLE_PATTERN value for HSS port hPort.  
    (n=hPort).  
  
#define IX_NPE_A_MSSG_HSS_CHAN_NUM_CHANS_WRITE  
    HSS Message ID command writes the HSSnC_NUM_CHANNELS value for HSS port hPort.  
    (n=hPort).  
  
#define IX_NPE_A_MSSG_HSS_CHAN_RX_BUF_ADDR_WRITE  
    HSS Message ID command writes the HSSnC_RX_BUF_ADDR value for HSS port hPort. (n=hPort).  
  
#define IX_NPE_A_MSSG_HSS_CHAN_RX_BUF_CFG_WRITE  
    HSS Message ID command writes the HSSnC_RX_BUF_SIZEB and HSSnC_RX_TRIG_PERIOD  
    values for HSS port hPort. (n=hPort).  
  
#define IX_NPE_A_MSSG_HSS_CHAN_TX_BLK_CFG_WRITE  
    HSS Message ID command writes the HSSnC_TX_BLK1_SIZEB, HSSnC_TX_BLK1_SIZEW,  
    HSSnC_TX_BLK2_SIZEB, and HSSnC_TX_BLK2_SIZEW values for HSS port hPort. (n=hPort).  
  
#define IX_NPE_A_MSSG_HSS_CHAN_TX_BUF_ADDR_WRITE  
    HSS Message ID command writes the HSSnC_TX_BUF_ADDR value for HSS port hPort. (n=hPort).  
  
#define IX_NPE_A_MSSG_HSS_CHAN_TX_BUF_SIZE_WRITE  
    HSS Message ID command writes the HSSnC_TX_BUF_SIZEN value for HSS port hPort. (n=hPort).  
  
#define IX_NPE_A_MSSG_HSS_PKT_PIPE_FLOW_ENABLE  
    HSS Message ID command triggers the NPE to reset internal status and enable the HssPacketized  
    operation for the flow specified by pPipe on the HSS port specified by hPort.
```

```

#define IX_NPE_A_MSSG_HSS_PKT_PIPE_FLOW_DISABLE
    HSS Message ID command triggers the NPE to disable the HssPacketized operation for the flow
    specified by pPipe on the HSS port specified by hPort.

#define IX_NPE_A_MSSG_HSS_PKT_NUM_PIPES_WRITE
    HSS Message ID command writes the HSSnP_NUM_PIPES value for HSS port hPort.(n=hPort).

#define IX_NPE_A_MSSG_HSS_PKT_PIPE_FIFO_SIZEW_WRITE
    HSS Message ID command writes the HSSnP_PIPEp_FIFO_SIZEW value for packet-pipe pPipe on
    HSS port hPort. (n=hPort, p=pPipe).

#define IX_NPE_A_MSSG_HSS_PKT_PIPE_HDLC_CFG_WRITE
    HSS Message ID command writes the HSSnP_PIPEp_HDLC_RXCFG and
    HSSnP_PIPEp_HDLC_TXCFG values for packet-pipe pPipe on HSS port hPort. (n=hPort,
    p=pPipe).

#define IX_NPE_A_MSSG_HSS_PKT_PIPE_IDLE_PATTERN_WRITE
    HSS Message ID command writes the HSSnP_PIPEp_IDLE_PATTERN value for packet-pipe pPipe
    on HSS port hPort. (n=hPort, p=pPipe).

#define IX_NPE_A_MSSG_HSS_PKT_PIPE_RX_SIZE_WRITE
    HSS Message ID command writes the HSSnP_PIPEp_RX_SIZEB value for packet-pipe pPipe on HSS
    port hPort. (n=hPort, p=pPipe).

#define IX_NPE_A_MSSG_HSS_PKT_PIPE_MODE_WRITE
    HSS Message ID command writes the HSSnP_PIPEp_MODE value for packet-pipe pPipe on HSS
    port hPort. (n=hPort, p=pPipe).

#define IX_NPE_A_RXDESCRIPTOR_STATUS_OFFSET
    ATM Descriptor structure offset for Receive Descriptor Status field.

#define IX_NPE_A_RXDESCRIPTOR_VCID_OFFSET
    ATM Descriptor structure offset for Receive Descriptor VC ID field.

#define IX_NPE_A_RXDESCRIPTOR_CURRMBUFSIZE_OFFSET
    ATM Descriptor structure offset for Receive Descriptor Current Mbuf Size field.

#define IX_NPE_A_RXDESCRIPTOR_ATMHEADER_OFFSET
    ATM Descriptor structure offset for Receive Descriptor ATM Header.

#define IX_NPE_A_RXDESCRIPTOR_CURRMBUFLEN_OFFSET
    ATM Descriptor structure offset for Receive Descriptor Current MBuf length.

#define IX_NPE_A_RXDESCRIPTOR_TIMELIMIT_OFFSET
#define IX_NPE_A_RXDESCRIPTOR_PCURRMBUFF_OFFSET
    ATM Descriptor structure offset for Receive Descriptor Current MBuf Pointer.

#define IX_NPE_A_RXDESCRIPTOR_PCURRMBUFDATA_OFFSET
    ATM Descriptor structure offset for Receive Descriptor Current MBuf Pointer.

#define IX_NPE_A_RXDESCRIPTOR_PNEXTMBUF_OFFSET

```



*ATM Descriptor structure offset for Receive Descriptor Next MBuf Pointer.*

**#define IX\_NPE\_A\_RXDESCRIPTOR\_TOTALLENGTH\_OFFSET**  
*ATM Descriptor structure offset for Receive Descriptor Total Length.*

**#define IX\_NPE\_A\_RXDESCRIPTOR\_AAL5CRCRESIDUE\_OFFSET**  
*ATM Descriptor structure offset for Receive Descriptor AAL5 CRC Residue.*

**#define IX\_NPE\_A\_RXDESCRIPTOR\_SIZE**  
*ATM Descriptor structure offset for Receive Descriptor Size.*

**#define IX\_NPE\_A\_TXDESCRIPTOR\_PORT\_OFFSET**  
*ATM Descriptor structure offset for Transmit Descriptor Port.*

**#define IX\_NPE\_A\_TXDESCRIPTOR\_RSVD\_OFFSET**  
*ATM Descriptor structure offset for Transmit Descriptor RSVD.*

**#define IX\_NPE\_A\_TXDESCRIPTOR\_CURRMBUFLEN\_OFFSET**  
*ATM Descriptor structure offset for Transmit Descriptor Current MBuf Length.*

**#define IX\_NPE\_A\_TXDESCRIPTOR\_ATMHEADER\_OFFSET**  
*ATM Descriptor structure offset for Transmit Descriptor ATM Header.*

**#define IX\_NPE\_A\_TXDESCRIPTOR\_PCURRMBUFF\_OFFSET**  
*ATM Descriptor structure offset for Transmit Descriptor Pointer to the current MBuf chain.*

**#define IX\_NPE\_A\_TXDESCRIPTOR\_PCURRMBUFDATA\_OFFSET**  
*ATM Descriptor structure offset for Transmit Descriptor Pointer to the current MBuf Data.*

**#define IX\_NPE\_A\_TXDESCRIPTOR\_PNEXTMBUF\_OFFSET**  
*ATM Descriptor structure offset for Transmit Descriptor Pointer to the Next MBuf chain.*

**#define IX\_NPE\_A\_TXDESCRIPTOR\_TOTALLENGTH\_OFFSET**  
*ATM Descriptor structure offset for Transmit Descriptor Total Length.*

**#define IX\_NPE\_A\_TXDESCRIPTOR\_AAL5CRCRESIDUE\_OFFSET**  
*ATM Descriptor structure offset for Transmit Descriptor AAL5 CRC Residue.*

**#define IX\_NPE\_A\_TXDESCRIPTOR\_SIZE**  
*ATM Descriptor structure offset for Transmit Descriptor Size.*

**#define IX\_NPE\_MPHYMULTIPORT**  
*Define this macro to enable MPHY mode.*

**#define IX\_NPE\_A\_TXDONE\_QUEUE\_HIGHWATERMARK**  
*The NPE reserves the High Watermark for its operation. But it must be set by the Xscale.*

**#define IX\_NPE\_A\_QMQ\_ATM\_TX\_DONE**  
*Queue ID for ATM Transmit Done queue.*

**#define IX\_NPE\_A\_QMQ\_ATM\_TX0**

*Queue ID for ATM transmit Queue in a single phy configuration.*

**#define IX\_NPE\_A\_QMQ\_ATM\_TXID\_MIN**  
*Queue Manager Queue ID for ATM transmit Queue with minimum number of queue.*

**#define IX\_NPE\_A\_QMQ\_ATM\_TXID\_MAX**  
*Queue Manager Queue ID for ATM transmit Queue with maximum number of queue.*

**#define IX\_NPE\_A\_QMQ\_ATM\_RX\_HI**  
*Queue Manager Queue ID for ATM Receive high Queue.*

**#define IX\_NPE\_A\_QMQ\_ATM\_RX\_LO**  
*Queue Manager Queue ID for ATM Receive low Queue.*

**#define IX\_NPE\_A\_QMQ\_ATM\_TX1**  
*Queue ID for ATM transmit Queue Multiplier from 1 to 11.*

**#define IX\_NPE\_A\_QMQ\_ATM\_TX2**

**#define IX\_NPE\_A\_QMQ\_ATM\_TX3**

**#define IX\_NPE\_A\_QMQ\_ATM\_TX4**

**#define IX\_NPE\_A\_QMQ\_ATM\_TX5**

**#define IX\_NPE\_A\_QMQ\_ATM\_TX6**

**#define IX\_NPE\_A\_QMQ\_ATM\_TX7**

**#define IX\_NPE\_A\_QMQ\_ATM\_TX8**

**#define IX\_NPE\_A\_QMQ\_ATM\_TX9**

**#define IX\_NPE\_A\_QMQ\_ATM\_TX10**

**#define IX\_NPE\_A\_QMQ\_ATM\_TX11**

**#define IX\_NPE\_A\_QMQ\_HSS0\_CHL\_RX\_TRIG**  
*Hardware QMgr Queue ID for HSS Port 0 Channelized Receive trigger.*

**#define IX\_NPE\_A\_QMQ\_HSS0\_PKT\_RX**  
*Hardware QMgr Queue ID for HSS Port 0 Packetized Receive.*

**#define IX\_NPE\_A\_QMQ\_HSS0\_PKT\_TX0**  
*Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 0.*

**#define IX\_NPE\_A\_QMQ\_HSS0\_PKT\_TX1**  
*Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 1.*

**#define IX\_NPE\_A\_QMQ\_HSS0\_PKT\_TX2**  
*Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 2.*

**#define IX\_NPE\_A\_QMQ\_HSS0\_PKT\_TX3**  
*Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 3.*

**#define IX\_NPE\_A\_QMQ\_HSS0\_PKT\_RX\_FREE0**  
*Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 0.*

**#define IX\_NPE\_A\_QMQ\_HSS0\_PKT\_RX\_FREE1**  
*Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 1.*

```

#define IX_NPE_A_QMQ_HSS0_PKT_RX_FREE2
    Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 2.

#define IX_NPE_A_QMQ_HSS0_PKT_RX_FREE3
    Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 3.

#define IX_NPE_A_QMQ_HSS0_PKT_TX_DONE
    Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit Done queue.

#define IX_NPE_A_QMQ_HSS1_CHL_RX_TRIG
    Hardware QMgr Queue ID for HSS Port 1 Channelized Receive trigger.

#define IX_NPE_A_QMQ_HSS1_PKT_RX
    Hardware QMgr Queue ID for HSS Port 1 Packetized Receive.

#define IX_NPE_A_QMQ_HSS1_PKT_TX0
    Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 0.

#define IX_NPE_A_QMQ_HSS1_PKT_TX1
    Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 1.

#define IX_NPE_A_QMQ_HSS1_PKT_TX2
    Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 2.

#define IX_NPE_A_QMQ_HSS1_PKT_TX3
    Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 3.

#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE0
    Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 0.

#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE1
    Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 1.

#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE2
    Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 2.

#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE3
    Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 3.

#define IX_NPE_A_QMQ_HSS1_PKT_TX_DONE
    Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit Done queue.

#define IX_NPE_A_QMQ_ATM_FREE_VC0
    Hardware QMgr Queue ID for ATM Free VC Queue.

#define IX_NPE_A_QMQ_ATM_FREE_VC1
#define IX_NPE_A_QMQ_ATM_FREE_VC2
#define IX_NPE_A_QMQ_ATM_FREE_VC3
#define IX_NPE_A_QMQ_ATM_FREE_VC4
#define IX_NPE_A_QMQ_ATM_FREE_VC5
#define IX_NPE_A_QMQ_ATM_FREE_VC6

```

```

#define IX_NPE_A_QMQ_ATM_FREE_VC7
#define IX_NPE_A_QMQ_ATM_FREE_VC8
#define IX_NPE_A_QMQ_ATM_FREE_VC9
#define IX_NPE_A_QMQ_ATM_FREE_VC10
#define IX_NPE_A_QMQ_ATM_FREE_VC11
#define IX_NPE_A_QMQ_ATM_FREE_VC12
#define IX_NPE_A_QMQ_ATM_FREE_VC13
#define IX_NPE_A_QMQ_ATM_FREE_VC14
#define IX_NPE_A_QMQ_ATM_FREE_VC15
#define IX_NPE_A_QMQ_ATM_FREE_VC16
#define IX_NPE_A_QMQ_ATM_FREE_VC17
#define IX_NPE_A_QMQ_ATM_FREE_VC18
#define IX_NPE_A_QMQ_ATM_FREE_VC19
#define IX_NPE_A_QMQ_ATM_FREE_VC20
#define IX_NPE_A_QMQ_ATM_FREE_VC21
#define IX_NPE_A_QMQ_ATM_FREE_VC22
#define IX_NPE_A_QMQ_ATM_FREE_VC23
#define IX_NPE_A_QMQ_ATM_FREE_VC24
#define IX_NPE_A_QMQ_ATM_FREE_VC25
#define IX_NPE_A_QMQ_ATM_FREE_VC26
#define IX_NPE_A_QMQ_ATM_FREE_VC27
#define IX_NPE_A_QMQ_ATM_FREE_VC28
#define IX_NPE_A_QMQ_ATM_FREE_VC29
#define IX_NPE_A_QMQ_ATM_FREE_VC30
#define IX_NPE_A_QMQ_ATM_FREE_VC31
#define IX_NPE_A_QMQ_ATM_RXFREE_MIN
    The minimum queue ID for FreeVC queue.

#define IX_NPE_A_QMQ_ATM_RXFREE_MAX
    The maximum queue ID for FreeVC queue.

#define IX_NPE_A_QMQ_OAM_FREE_VC
    OAM Rx Free queue ID.

#define IX_NPE_A_CHAIN_DESC_COUNT_MAX
    Maximum number of chained MBufs that can be chained together.

#define GFC_MASK
    Mask to access GFC.

#define IX_NPE_A_ATMCELLHEADER_GFC_GET(header)
    return GFC from ATM cell header

#define IX_NPE_A_ATMCELLHEADER_GFC_SET(header, gfc)
    set GFC into ATM cell header

#define VPI_MASK
    Mask to access VPI.

#define IX_NPE_A_ATMCELLHEADER_VPI_GET(header)

```

```

    return VPI from ATM cell header

#define IX_NPE_A_ATMCELLHEADER_VPI_SET(header, vpi)
    set VPI into ATM cell header

#define VCI_MASK
    Mask to access VCI.

#define IX_NPE_A_ATMCELLHEADER_VCI_GET(header)
    return VCI from ATM cell header

#define IX_NPE_A_ATMCELLHEADER_VCI_SET(header, vci)
    set VCI into ATM cell header

#define PTI_MASK
    Mask to access PTI.

#define IX_NPE_A_ATMCELLHEADER_PTI_GET(header)
    return PTI from ATM cell header

#define IX_NPE_A_ATMCELLHEADER_PTI_SET(header, pti)
    set PTI into ATM cell header

#define CLP_MASK
    Mask to access CLP.

#define IX_NPE_A_ATMCELLHEADER_CLP_GET(header)
    return CLP from ATM cell header

#define IX_NPE_A_ATMCELLHEADER_CLP_SET(header, clp)
    set CLP into ATM cell header

#define STATUS_MASK
    Mask to access the rxBitField status.

#define IX_NPE_A_RXBITFIELD_STATUS_GET(rxbitfield)
    return the rxBitField status

#define IX_NPE_A_RXBITFIELD_STATUS_SET(rxbitfield, status)
    set the rxBitField status

#define PORT_MASK
    Mask to access the rxBitField port.

#define IX_NPE_A_RXBITFIELD_PORT_GET(rxbitfield)
    return the rxBitField port

#define IX_NPE_A_RXBITFIELD_PORT_SET(rxbitfield, port)
    set the rxBitField port

#define VCID_MASK

```

*Mask to access the rxBitField vcid.*

```
#define IX_NPE_A_RXBITFIELD_VCID_GET(rxbitfield)  
    return the rxBitField vcid
```

```
#define IX_NPE_A_RXBITFIELD_VCID_SET(rxbitfield, vcid)  
    set the rxBitField vcid
```

```
#define CURRMBUFSIZE_MASK  
    Mask to access the rxBitField mbuf size.
```

```
#define IX_NPE_A_RXBITFIELD_CURRMBUFSIZE_GET(rxbitfield)  
    return the rxBitField mbuf size
```

```
#define IX_NPE_A_RXBITFIELD_CURRMBUFSIZE_SET(rxbitfield, currmbufsize)  
    set the rxBitField mbuf size
```

## Enumerations

```
enum IxNpeA_AalType {  
    IX_NPE_A_AAL_TYPE_INVALID,  
    IX_NPE_A_AAL_TYPE_0_48,  
    IX_NPE_A_AAL_TYPE_0_52,  
    IX_NPE_A_AAL_TYPE_5,  
    IX_NPE_A_AAL_TYPE_OAM  
}  
NPE-A AAL Type.
```

```
enum IxNpeA_PayloadFormat {  
    IX_NPE_A_52_BYTE_PAYLOAD,  
    IX_NPE_A_48_BYTE_PAYLOAD  
}  
NPE-A Payload format 52-bytes & 48-bytes.
```

---

## Detailed Description

The Public API for the IXP425 NPE-A.

---

## Define Documentation

```
#define CLP_MASK
```

Mask to access CLP.

Definition at line **920** of file **IxNpeA.h**.

```
#define CURRMBUFSIZE_MASK
```

Mask to access the rxBitField mbuf size.

Definition at line **999** of file **IxNpeA.h**.

```
#define GFC_MASK
```

Mask to access GFC.

Definition at line **864** of file **IxNpeA.h**.

```
#define IX_NPE_A_ATMCELLHEADER_CLP_GET ( header )
```

return CLP from ATM cell header

Definition at line **923** of file **IxNpeA.h**.

```
#define IX_NPE_A_ATMCELLHEADER_CLP_SET ( header,  
                                         clp    )
```

set CLP into ATM cell header

Definition at line **927** of file **IxNpeA.h**.

```
#define IX_NPE_A_ATMCELLHEADER_GFC_GET ( header )
```

return GFC from ATM cell header

Definition at line **867** of file **IxNpeA.h**.

```
#define IX_NPE_A_ATMCELLHEADER_GFC_SET ( header,  
                                         gfc    )
```

set GFC into ATM cell header

Definition at line **871** of file **IxNpeA.h**.

```
#define IX_NPE_A_ATMCELLHEADER_PTI_GET ( header )
```

return PTI from ATM cell header

Definition at line **909** of file **IxNpeA.h**.

```
#define IX_NPE_A_ATMCELLHEADER_PTI_SET ( header,  
                                         pti      )
```

set PTI into ATM cell header

Definition at line **913** of file **IxNpeA.h**.

```
#define IX_NPE_A_ATMCELLHEADER_VCI_GET ( header )
```

return VCI from ATM cell header

Definition at line **895** of file **IxNpeA.h**.

```
#define IX_NPE_A_ATMCELLHEADER_VCI_SET ( header,  
                                         vci      )
```

set VCI into ATM cell header

Definition at line **899** of file **IxNpeA.h**.

```
#define IX_NPE_A_ATMCELLHEADER_VPI_GET ( header )
```

return VPI from ATM cell header

Definition at line **881** of file **IxNpeA.h**.

```
#define IX_NPE_A_ATMCELLHEADER_VPI_SET ( header,  
                                         vpi      )
```

set VPI into ATM cell header

Definition at line **885** of file **IxNpeA.h**.

```
#define IX_NPE_A_CHAIN_DESC_COUNT_MAX
```

Maximum number of chained MBufs that can be chained together.

Definition at line **837** of file **IxNpeA.h**.



```
#define IX_NPE_A_MSSG_ATM_RX_DISABLE
```

ATM Message ID command triggers the NPE to disable processing for this VC.

This command will be ignored for a VC already disabled

Definition at line **138** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_ATM_RX_ENABLE
```

ATM Message ID command triggers the NPE to process any received cells for this VC according to the VC Lookup Table.

Re-issuing this command with different contents for a VC that is not disabled will cause unpredictable behavior.

Definition at line **128** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_ATM_STATUS_READ
```

ATM Message ID command to read the ATM status. The data is returned via a response message.

Definition at line **146** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_ATM_TX_DISABLE
```

ATM Message ID command triggers the NPE to disable processing on this port.

This command will be ignored for a port already disabled

Definition at line **117** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_ATM_TX_ENABLE
```

ATM Message ID command triggers the NPE to re-enable processing of any entries on the TxVcQ for this port.

This command will be ignored for a port already enabled

Definition at line **107** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_ATM_UTOPIA_CONFIG_LOAD
```

ATM Message ID command triggers the NPE to copy the Utopia Configuration Table to the Utopia coprocessor.

Definition at line **81** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_ATM_UTOPIA_CONFIG_WRITE
```

ATM Message ID command to write the data to the offset in the Utopia Configuration Table.

Definition at line **73** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_ATM_UTOPIA_STATUS_READ
```

ATM Message ID command to read the Utopia Status Table at the specified offset.

Definition at line **97** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_ATM_UTOPIA_STATUS_UPLOAD
```

ATM Message ID command triggers the NPE to read-back the Utopia status registers and update the Utopia Status Table.

Definition at line **89** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_CHAN_FLOW_DISABLE
```

HSS Message ID command triggers the NPE to disable the HssChannelized operation on the HSS port specified by hPort.

Definition at line **191** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_CHAN_FLOW_ENABLE
```

HSS Message ID command triggers the NPE to reset internal status and enable the HssChannelized operation on the HSS port specified by hPort.

Definition at line **183** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_CHAN_IDLE_PATTERN_WRITE
```

HSS Message ID command writes the HSSnC\_IDLE\_PATTERN value for HSS port hPort. (n=hPort).

Definition at line **199** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_CHAN_NUM_CHANS_WRITE
```

HSS Message ID command writes the HSSnC\_NUM\_CHANNELS value for HSS port hPort. (n=hPort).

Definition at line **207** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_CHAN_RX_BUF_ADDR_WRITE
```

HSS Message ID command writes the HSSnC\_RX\_BUF\_ADDR value for HSS port hPort. (n=hPort).

Definition at line **215** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_CHAN_RX_BUF_CFG_WRITE
```

HSS Message ID command writes the HSSnC\_RX\_BUF\_SIZEB and HSSnC\_RX\_TRIG\_PERIOD values for HSS port hPort. (n=hPort).

Definition at line **223** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_CHAN_TX_BLK_CFG_WRITE
```

HSS Message ID command writes the HSSnC\_TX\_BLK1\_SIZEB, HSSnC\_TX\_BLK1\_SIZEW, HSSnC\_TX\_BLK2\_SIZEB, and HSSnC\_TX\_BLK2\_SIZEW values for HSS port hPort. (n=hPort).

Definition at line **232** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_CHAN_TX_BUF_ADDR_WRITE
```

HSS Message ID command writes the HSSnC\_TX\_BUF\_ADDR value for HSS port hPort. (n=hPort).

Definition at line **239** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_CHAN_TX_BUF_SIZE_WRITE
```

HSS Message ID command writes the HSSnC\_TX\_BUF\_SIZEN value for HSS port hPort. (n=hPort).

Definition at line **247** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PKT_NUM_PIPES_WRITE
```

HSS Message ID command writes the HSSnP\_NUM\_PIPES value for HSS port hPort.(n=hPort).

Definition at line **270** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PKT_PIPE_FIFO_SIZEW_WRITE
```

HSS Message ID command writes the HSSnP\_PIPEp\_FIFO\_SIZEW value for packet-pipe pPipe on HSS port hPort. (n=hPort, p=pPipe).

Definition at line **278** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PKT_PIPE_FLOW_DISABLE
```

HSS Message ID command triggers the NPE to disable the HssPacketized operation for the flow specified by pPipe on the HSS port specified by hPort.

Definition at line **263** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PKT_PIPE_FLOW_ENABLE
```

HSS Message ID command triggers the NPE to reset internal status and enable the HssPacketized operation for the flow specified by pPipe on the HSS port specified by hPort.

Definition at line **256** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PKT_PIPE_HDLC_CFG_WRITE
```

HSS Message ID command writes the HSSnP\_PIPEp\_HDLC\_RXCFG and HSSnP\_PIPEp\_HDLC\_TXCFG values for packet-pipe pPipe on HSS port hPort. (n=hPort, p=pPipe).

Definition at line **287** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PKT_PIPE_IDLE_PATTERN_WRITE
```

HSS Message ID command writes the HSSnP\_PIPEp\_IDLE\_PATTERN value for packet-pipe pPipe on HSS port hPort. (n=hPort, p=pPipe).

Definition at line **295** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PKT_PIPE_MODE_WRITE
```

HSS Message ID command writes the HSSnP\_PIPEp\_MODE value for packet-pipe pPipe on HSS port hPort. (n=hPort, p=pPipe).

Definition at line **311** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PKT_PIPE_RX_SIZE_WRITE
```

HSS Message ID command writes the HSSnP\_PIPEp\_RXSIZEB value for packet-pipe pPipe on HSS port hPort. (n=hPort, p=pPipe).

Definition at line **303** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PORT_CONFIG_LOAD
```

HSS Message ID command triggers the NPE to copy the contents of the HSS Configuration Table to the appropriate configuration registers in the HSS coprocessor for the port specified by hPort.

Definition at line **167** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PORT_CONFIG_WRITE
```

HSS Message ID command writes the ConfigWord value to the location in the HSS\_CONFIG\_TABLE specified by offset for HSS port hPort.

Definition at line **158** of file **IxNpeA.h**.

```
#define IX_NPE_A_MSSG_HSS_PORT_ERROR_READ
```

HSS Message ID command triggers the NPE to return an HssErrorReadResponse message for HSS port hPort.

Definition at line **175** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_ATM_FREE_VC0
```

Hardware QMgr Queue ID for ATM Free VC Queue.

There are 32 Hardware QMgr Queue ID; from IX\_NPE\_A\_QMQ\_ATM\_FREE\_VC1 to IX\_NPE\_A\_QMQ\_ATM\_FREE\_VC30

Definition at line **775** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_ATM_RX_HI
```

Queue Manager Queue ID for ATM Receive high Queue.

Definition at line **591** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_ATM_RX_LO
```

Queue Manager Queue ID for ATM Receive low Queue.

Definition at line **592** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_ATM_RXFREE_MAX
```

The maximum queue ID for FreeVC queue.

Definition at line **820** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_ATM_RXFREE_MIN
```

The minimum queue ID for FreeVC queue.

Definition at line **813** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_ATM_TX0
```

Queue ID for ATM transmit Queue in a single phy configuration.

Definition at line **546** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_ATM_TX1
```

Queue ID for ATM transmit Queue Multiplier from 1 to 11.

Definition at line **578** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_ATM_TX_DONE
```

Queue ID for ATM Transmit Done queue.

Definition at line **539** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_ATM_TXID_MAX
```

Queue Manager Queue ID for ATM transmit Queue with maximum number of queue.

Definition at line **590** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_ATM_TXID_MIN
```

Queue Manager Queue ID for ATM transmit Queue with minimum number of queue.

Definition at line **589** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_CHL_RX_TRIG
```

Hardware QMgr Queue ID for HSS Port 0 Channelized Receive trigger.

Definition at line **609** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_PKT_RX
```

Hardware QMgr Queue ID for HSS Port 0 Packetized Receive.

Definition at line **616** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_PKT_RX_FREE0
```

Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 0.

Definition at line **651** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_PKT_RX_FREE1
```

Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 1.

Definition at line **658** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_PKT_RX_FREE2
```

Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 2.

Definition at line **665** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_PKT_RX_FREE3
```

Hardware QMgr Queue ID for HSS Port 0 Packetized Receive Free queue 3.

Definition at line **672** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_PKT_TX0
```

Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 0.

Definition at line **623** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_PKT_TX1
```

Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 1.

Definition at line **630** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_PKT_TX2
```

Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 2.

Definition at line **637** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_PKT_TX3
```

Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit queue 3.

Definition at line **644** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS0_PKT_TX_DONE
```

Hardware QMgr Queue ID for HSS Port 0 Packetized Transmit Done queue.

Definition at line **679** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS1_CHL_RX_TRIG
```

Hardware QMgr Queue ID for HSS Port 1 Channelized Receive trigger.

Definition at line **691** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS1_PKT_RX
```

Hardware QMgr Queue ID for HSS Port 1 Packetized Receive.

Definition at line **698** of file **IxNpeA.h**.



```
#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE0
```

Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 0.

Definition at line **733** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE1
```

Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 1.

Definition at line **740** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE2
```

Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 2.

Definition at line **747** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS1_PKT_RX_FREE3
```

Hardware QMgr Queue ID for HSS Port 1 Packetized Receive Free queue 3.

Definition at line **754** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS1_PKT_TX0
```

Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 0.

Definition at line **705** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS1_PKT_TX1
```

Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 1.

Definition at line **712** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS1_PKT_TX2
```

Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 2.

Definition at line **719** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS1_PKT_TX3
```

Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit queue 3.

Definition at line **726** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_HSS1_PKT_TX_DONE
```

Hardware QMgr Queue ID for HSS Port 1 Packetized Transmit Done queue.

Definition at line **761** of file **IxNpeA.h**.

```
#define IX_NPE_A_QMQ_OAM_FREE_VC
```

OAM Rx Free queue ID.

Definition at line **827** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXBITFIELD_CURRMBUFSIZE_GET ( rxbitfield )
```

return the rxBitField mbuf size

Definition at line **1002** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXBITFIELD_CURRMBUFSIZE_SET ( rxbitfield,  
                                                currmbufsize )
```

set the rxBitField mbuf size

Definition at line **1006** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXBITFIELD_PORT_GET ( rxbitfield )
```

return the rxBitField port

Definition at line **974** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXBITFIELD_PORT_SET ( rxbitfield,  
                                       port      )
```

set the rxBitField port

Definition at line **978** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXBITFIELD_STATUS_GET ( rxbitfield )
```

return the rxBitField status

Definition at line **960** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXBITFIELD_STATUS_SET ( rxbitfield,  
                                         status      )
```

set the rxBitField status

Definition at line **964** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXBITFIELD_VCID_GET ( rxbitfield )
```

return the rxBitField vcId

Definition at line **988** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXBITFIELD_VCID_SET ( rxbitfield,  
                                       vcid      )
```

set the rxBitField vcId

Definition at line **992** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_AAL5CRCRESIDUE_OFFSET
```

ATM Descriptor structure offset for Receive Descriptor AAL5 CRC Residue.

Current CRC value for a PDU

Definition at line **419** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_ATMHEADER_OFFSET
```

ATM Descriptor structure offset for Receive Descriptor ATM Header.

Definition at line **354** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_CURRMBUFLEN_OFFSET
```

ATM Descriptor structure offset for Receive Descriptor Current MBuf length.

RX – Initialized to zero. The NPE updates this field as each cell is received and zeroes it with every new mbuf for chaining. Will not be bigger than currBbufSize.

Definition at line **365** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_CURRMBUFSIZE_OFFSET
```

ATM Descriptor structure offset for Receive Descriptor Current Mbuf Size field.

Number of bytes the current mbuf data buffer can hold

Definition at line **347** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_PCURRMBUFDATA_OFFSET
```

ATM Descriptor structure offset for Receive Descriptor Current MBuf Pointer.

Pointer to the next byte to be read or next free location to be written.

Definition at line **392** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_PCURRMBUFF_OFFSET
```

ATM Descriptor structure offset for Receive Descriptor Current MBuf Pointer.

The current mbuf pointer of a chain of mbufs.

Definition at line **383** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_PNEXTMBUF_OFFSET
```

ATM Descriptor structure offset for Receive Descriptor Next MBuf Pointer.

Pointer to the next MBuf in a chain of MBufs.

Definition at line **401** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_SIZE
```

ATM Descriptor structure offset for Receive Descriptor Size.

The size of the Receive descriptor

Definition at line **428** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_STATUS_OFFSET
```

ATM Descriptor structure offset for Receive Descriptor Status field.

It is used for descriptor error reporting.

Definition at line **328** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_TOTALLENGTH_OFFSET
```

ATM Descriptor structure offset for Receive Descriptor Total Length.

Total number of bytes written to the chain of MBufs by the NPE

Definition at line **410** of file **IxNpeA.h**.

```
#define IX_NPE_A_RXDESCRIPTOR_VCID_OFFSET
```

ATM Descriptor structure offset for Receive Descriptor VC ID field.

It is used to hold an identifier number for this VC

Definition at line **337** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDESCRIPTOR_AAL5CRCRESIDUE_OFFSET
```

ATM Descriptor structure offset for Transmit Descriptor AAL5 CRC Residue.

Current CRC value for a PDU

Definition at line **502** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDESCRIPTOR_ATMHEADER_OFFSET
```

ATM Descriptor structure offset for Transmit Descriptor ATM Header.

Definition at line **461** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDESCRIPTOR_CURRMBUFLLEN_OFFSET
```

ATM Descriptor structure offset for Transmit Descriptor Current MBuf Length.

TX – Initialized by the XScale to the number of bytes in the current MBuf data buffer. The NPE decrements this field for every transmitted cell. Thus, when the NPE writes a descriptor the TxDone queue, this field will equal zero.

Definition at line **455** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDESCRIPTOR_PCURRMBUFDATA_OFFSET
```

ATM Descriptor structure offset for Transmit Descriptor Pointer to the current MBuf Data.

Pointer to the next byte to be read or next free location to be written.

Definition at line **477** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDESCRIPTOR_PCURRMBUFF_OFFSET
```

ATM Descriptor structure offset for Transmit Descriptor Pointer to the current MBuf chain.

Definition at line **468** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDESCRIPTOR_PNEXTMBUF_OFFSET
```

ATM Descriptor structure offset for Transmit Descriptor Pointer to the Next MBuf chain.

Definition at line **484** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDESCRIPTOR_PORT_OFFSET
```

ATM Descriptor structure offset for Transmit Descriptor Port.

Port identifier.

Definition at line **437** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDESCRIPTOR_RSVD_OFFSET
```

ATM Descriptor structure offset for Transmit Descriptor RSVD.

Definition at line **444** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDESCRIPTOR_SIZE
```

ATM Descriptor structure offset for Transmit Descriptor Size.

Definition at line **509** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDESCRIPTOR_TOTALLENGTH_OFFSET
```

ATM Descriptor structure offset for Transmit Descriptor Total Length.

Total number of bytes written to the chain of MBufs by the NPE

Definition at line **493** of file **IxNpeA.h**.

```
#define IX_NPE_A_TXDONE_QUEUE_HIGHWATERMARK
```

The NPE reserves the High Watermark for its operation. But it must be set by the Xscale.

Definition at line **532** of file **IxNpeA.h**.

```
#define IX_NPE_MPHYMULTIPOINT
```

Define this macro to enable MPHY mode.

Definition at line **516** of file **IxNpeA.h**.

```
#define PORT_MASK
```

Mask to access the rxBitField port.

Definition at line **971** of file **IxNpeA.h**.

```
#define PTI_MASK
```

Mask to access PTI.

Definition at line **906** of file **IxNpeA.h**.

```
#define STATUS_MASK
```

Mask to access the rxBitField status.

Definition at line **957** of file **IxNpeA.h**.

```
#define VCI_MASK
```

Mask to access VCI.

Definition at line **892** of file **IxNpeA.h**.

```
#define VCID_MASK
```

Mask to access the rxBitField vcId.

Definition at line **985** of file **IxNpeA.h**.

```
#define VPI_MASK
```

Mask to access VPI.

Definition at line **878** of file **IxNpeA.h**.

---

## Enumeration Type Documentation

```
enum IxNpeA_AalType
```

NPE–A AAL Type.

### ***Enumeration values:***

<i>IX_NPE_A_AAL_TYPE_INVALID</i>	Invalid AAL type.
<i>IX_NPE_A_AAL_TYPE_0_48</i>	AAL0 – 48 byte.
<i>IX_NPE_A_AAL_TYPE_0_52</i>	AAL0 – 52 byte.
<i>IX_NPE_A_AAL_TYPE_5</i>	AAL5.
<i>IX_NPE_A_AAL_TYPE_OAM</i>	OAM.

Definition at line **1059** of file **IxNpeA.h**.

```
enum IxNpeA_PayloadFormat
```

NPE–A Payload format 52–bytes & 48–bytes.

### ***Enumeration values:***

<i>IX_NPE_A_52_BYTE_PAYLOAD</i>	52 byte payload
<i>IX_NPE_A_48_BYTE_PAYLOAD</i>	



48 byte  
payload

Definition at line **1071** of file **IxNpeA.h**.

# IXP425 NPE–Downloader (IxNpeDI) API

The Public API for the IXP425 NPE Downloader.

## Modules

### IXP425 NPE Image ID

#### Definition

*Definition of NPE Image ID to be passed to **ixNpeDIInitAndStart()** as input of type **UINT32** which has the following fields format:.*

## Data Structures

struct **IxNpeDIImageId**

*Image Id to identify each image contained in an image library.*

## Defines

#define **IX\_NPEDL\_PARAM\_ERR**

*NpeDl function return value for a parameter error.*

#define **IX\_NPEDL\_RESOURCE\_ERR**

*NpeDl function return value for a resource error.*

#define **IX\_NPEDL\_CRITICAL\_NPE\_ERR**

*NpeDl function return value for a Critical NPE error occuring during download. Assume NPE is left in unstable condition if this value is returned.*

#define **IX\_NPEDL\_CRITICAL\_MICROCODE\_ERR**

*NpeDl function return value for a Critical Microcode error discovered during download. Assume NPE is left in unstable condition if this value is returned.*

#define **ixNpeDIMicrocodeImageOverride(x)**

*Map old terminology that uses term "image" to new term "image library".*

#define **IxNpeDIVersionId**

*Map old terminology that uses term "version" to new term "image".*

#define **ixNpeDIVersionDownload(x, y)**

*Map old terminology that uses term "version" to new term "image".*

#define **ixNpeDIAvailableVersionsCountGet(x)**

*Map old terminology that uses term "version" to new term "image".*

#define **ixNpeDIAvailableVersionsListGet(x, y)**

*Map old terminology that uses term "version" to new term "image".*

**#define IxNpeDILoadedVersionGet(x, y)**  
*Map old terminology that uses term "version" to new term "image".*

**#define clientImage**  
*Map old terminology that uses term "image" to new term "image library".*

**#define versionIdPtr**  
*Map old terminology that uses term "version" to new term "image".*

**#define numVersionsPtr**  
*Map old terminology that uses term "version" to new term "image".*

**#define versionIdListPtr**  
*Map old terminology that uses term "version" to new term "image".*

**#define IxNpeDIBuildId**  
*Map old terminology that uses term "buildId" to new term "functionalityId".*

**#define buildId**  
*Map old terminology that uses term "buildId" to new term "functionalityId".*

**#define IX\_NPEDL\_MicrocodeImage**  
*Map old terminology that uses term "image" to new term "image library".*

## Typedefs

**typedef UINT8 IxNpeDIFunctionalityId**  
*Used to make up Functionality ID field of Image Id.*

**typedef UINT8 IxNpeDIMajor**  
*Used to make up Major Release field of Image Id.*

**typedef UINT8 IxNpeDIMinor**  
*Used to make up Minor Revision field of Image Id.*

## Enumerations

**enum IxNpeDINpeId {**  
    **IX\_NPEDL\_NPEID\_NPEA,**  
    **IX\_NPEDL\_NPEID\_NPEB,**  
    **IX\_NPEDL\_NPEID\_NPEC,**  
    **IX\_NPEDL\_NPEID\_MAX**  
**}**  
*NpeId numbers to identify NPE A, B or C.*

## Functions

- PUBLIC**  
**IX\_STATUS ixNpeDlNpeInitAndStart** (UINT32 npeImageId)  
*Stop, reset, download microcode (firmware) and finally start NPE.*
- PUBLIC**  
**IX\_STATUS ixNpeDlCustomImageNpeInitAndStart** (UINT32 \*imageLibrary, UINT32 npeImageId)  
*Stop, reset, download microcode (firmware) and finally start NPE.*
- PUBLIC**  
**IX\_STATUS ixNpeDlLoadedImageFunctionalityGet** (IxNpeDlNpeId npeId, UINT8 \*functionalityId)  
*Gets the functionality of the image last loaded on a particular NPE.*
- IX\_STATUS ixNpeDlMicrocodeImageLibraryOverride** (UINT32 \*clientImageLibrary)  
*This instructs NPE Downloader to use client-supplied microcode image library.*
- PUBLIC**  
**IX\_STATUS ixNpeDlImageDownload** (IxNpeDlImageId \*imageIdPtr, BOOL verify)  
*Stop, reset, download microcode and finally start NPE.*
- PUBLIC**  
**IX\_STATUS ixNpeDlAvailableImagesCountGet** (UINT32 \*numImagesPtr)  
*Get the number of Images available in a microcode image library.*
- PUBLIC ixNpeDlAvailableImagesListGet** (IxNpeDlImageId \*imageIdListPtr, UINT32  
**IX\_STATUS \*listSizePtr**)  
*Get a list of the images available in a microcode image library.*
- PUBLIC**  
**IX\_STATUS ixNpeDlLoadedImageGet** (IxNpeDlNpeId npeId, IxNpeDlImageId \*imageIdPtr)  
*Gets the Id of the image currently loaded on a particular NPE.*
- PUBLIC ixNpeDlLatestImageGet** (IxNpeDlNpeId npeId, IxNpeDlFunctionalityId  
**IX\_STATUS functionalityId, IxNpeDlImageId \*imageIdPtr**)  
*This instructs NPE Downloader to get Id of the latest version for an image that is specified by the client.*
- PUBLIC**  
**IX\_STATUS ixNpeDlNpeStopAndReset** (IxNpeDlNpeId npeId)  
*Stops and Resets an NPE.*
- PUBLIC**  
**IX\_STATUS ixNpeDlNpeExecutionStart** (IxNpeDlNpeId npeId)  
*Starts code execution on a NPE.*
- PUBLIC**  
**IX\_STATUS ixNpeDlNpeExecutionStop** (IxNpeDlNpeId npeId)

*Stops code execution on a NPE.*

IX\_STATUS **ixNpeDIUnload** (void)

*This function will uninitialise the IxNpeDI component.*

PUBLIC void **ixNpeDIStatsShow** (void)

*This function will display run-time statistics from the IxNpeDI component.*

PUBLIC void **ixNpeDIStatsReset** (void)

*This function will reset the statistics of the IxNpeDI component.*

---

## Detailed Description

The Public API for the IXP425 NPE Downloader.

---

## Define Documentation

```
#define buildId
```

Map old terminology that uses term "buildId" to new term "functionalityId".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDIInitAndStart** for more information.

Definition at line **562** of file **IxNpeDI.h**.

```
#define clientImage
```

Map old terminology that uses term "image" to new term "image library".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDIInitAndStart** for more information.

Definition at line **502** of file **IxNpeDI.h**.

```
#define IX_NPEDL_CRITICAL_MICROCODE_ERR
```

NpeDI function return value for a Critical Microcode error discovered during download. Assume NPE is left in unstable condition if this value is returned.

Definition at line **100** of file **IxNpeDI.h**.

```
#define IX_NPEDL_CRITICAL_NPE_ERR
```

NpeDl function return value for a Critical NPE error occurring during download. Assume NPE is left in unstable condition if this value is returned.

Definition at line **91** of file **IxNpeDl.h**.

```
#define IX_NPEDL_MicrocodeImage
```

Map old terminology that uses term "image" to new term "image library".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **574** of file **IxNpeDl.h**.

```
#define IX_NPEDL_PARAM_ERR
```

NpeDl function return value for a parameter error.

Definition at line **75** of file **IxNpeDl.h**.

```
#define IX_NPEDL_RESOURCE_ERR
```

NpeDl function return value for a resource error.

Definition at line **82** of file **IxNpeDl.h**.

```
#define ixNpeDlAvailableVersionsCountGet ( x )
```

Map old terminology that uses term "version" to new term "image".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **466** of file **IxNpeDl.h**.

```
#define ixNpeDlAvailableVersionsListGet ( x,  
                                         y )
```

Map old terminology that uses term "version" to new term "image".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDIInitAndStart** for more information.

Definition at line **478** of file **IxNpeDI.h**.

```
#define IxNpeDIBuildId
```

Map old terminology that uses term "buildId" to new term "functionalityId".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDIInitAndStart** for more information.

Definition at line **550** of file **IxNpeDI.h**.

```
#define ixNpeDILoadedVersionGet ( x,  
                                y )
```

Map old terminology that uses term "version" to new term "image".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDIInitAndStart** for more information.

Definition at line **490** of file **IxNpeDI.h**.

```
#define ixNpeDIMicrocodeImageOverride ( x )
```

Map old terminology that uses term "image" to new term "image library".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDIInitAndStart** for more information.

Definition at line **430** of file **IxNpeDI.h**.

```
#define ixNpeDIVersionDownload ( x,  
                                y )
```

Map old terminology that uses term "version" to new term "image".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDIInitAndStart** for more information.

Definition at line **454** of file **IxNpeDL.h**.

```
#define IxNpeDLVersionId
```

Map old terminology that uses term "version" to new term "image".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDLNpeInitAndStart** for more information.

Definition at line **442** of file **IxNpeDL.h**.

```
#define numVersionsPtr
```

Map old terminology that uses term "version" to new term "image".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDLNpeInitAndStart** for more information.

Definition at line **526** of file **IxNpeDL.h**.

```
#define versionIdListPtr
```

Map old terminology that uses term "version" to new term "image".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDLNpeInitAndStart** for more information.

Definition at line **538** of file **IxNpeDL.h**.

```
#define versionIdPtr
```

Map old terminology that uses term "version" to new term "image".

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDLNpeInitAndStart** for more information.

Definition at line **514** of file **IxNpeDL.h**.

---



# Typedef Documentation

## IxNpeDlFunctionalityId

Used to make up Functionality ID field of Image Id.

**Warning:**

**THIS typedef HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **588** of file **IxNpeDl.h**.

## IxNpeDlMajor

Used to make up Major Release field of Image Id.

**Warning:**

**THIS typedef HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **598** of file **IxNpeDl.h**.

## IxNpeDlMinor

Used to make up Minor Revision field of Image Id.

**Warning:**

**THIS typedef HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **608** of file **IxNpeDl.h**.

---

# Enumeration Type Documentation

## enum IxNpeDlNpeId

NpeId numbers to identify NPE A, B or C.

**Note:**

- In this context, for IXP425 Silicon (B0):
- ◊ NPE–A has HDLC, HSS, AAL and UTOPIA Coprocessors.
  - ◊ NPE–B has Ethernet Coprocessor.
  - ◊ NPE–C has Ethernet, AES, DES and HASH

Coprocessors.  
◇ IXP425 Product Line have different combinations of coprocessors.

**Enumeration values:**

*IX\_NPEDL\_NPEID\_NPEA* Identifies NPE A.  
*IX\_NPEDL\_NPEID\_NPEB* Identifies NPE B.  
*IX\_NPEDL\_NPEID\_NPEC* Identifies NPE C.  
*IX\_NPEDL\_NPEID\_MAX* Total Number of NPEs.

Definition at line **623** of file **IxNpeDI.h**.

---

## Function Documentation

```
PUBLIC IX_STATUS ixNpeDIAvailableImagesCountGet ( UINT32 * numImagesPtr )
```

Get the number of Images available in a microcode image library.

**Parameters:**

*UINT32\** [out] *numImagesPtr* – A pointer to the number of images in the image library.

Gets the number of images available in the microcode image library. Then returns this in a variable pointed to by *numImagesPtr*.

**Warning:**

**THIS FUNCTION HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDINpeInitAndStart** and **ixNpeDICustomImageNpeInitAndStart**.

**Precondition:**

◇ The Client should declare the variable to which *numImagesPtr* points

**Postcondition:**

**Returns:**

◇ **IX\_SUCCESS** if the operation was successful  
◇ **IX\_NPEDL\_PARAM\_ERR** if a parameter error occurred  
◇ **IX\_FAIL** otherwise

```
PUBLIC IX_STATUS ixNpeDIAvailableImagesListGet ( IxNpeDIImageId * imageIdListPtr,  
                                                UINT32 * listSizePtr  
                                                )
```

Get a list of the images available in a microcode image library.

**Parameters:**

*IxNpeDIImageId\**

	[out] <i>imageIdListPtr</i> – Array to contain list of image Ids (memory allocated by Client).
<i>UINT32*</i>	[inout] <i>listSizePtr</i> – As an input, this param should point to the max number of Ids the <i>imageIdListPtr</i> array can hold. NpeDI will replace the input value of this parameter with the number of image Ids actually filled into the array before returning.

Finds list of images available in the microcode image library. Fills these into the array pointed to by *imageIdListPtr*

**Warning:**

**THIS FUNCTION HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDINpeInitAndStart** and **ixNpeDICustomImageNpeInitAndStart**.

**Precondition:**

- ◇ The Client should declare the variable to which *numImagesPtr* points
- ◇ The Client should create an array (*imageIdListPtr*) large enough to contain all the image Ids in the image library

**Postcondition:**

**Returns:**

- ◇ **IX\_SUCCESS** if the operation was successful
- ◇ **IX\_NPEDL\_PARAM\_ERR** if a parameter error occurred
- ◇ **IX\_FAIL** otherwise

```
PUBLIC IX_STATUS ixNpeDICustomImageNpeInitAndStart ( UINT32 * imageLibrary,
                                                    UINT32 npeImageId
                                                    )
```

Stop, reset, download microcode (firmware) and finally start NPE.

**Parameters:**

*UINT32* [in] *imageId* – Id of the microcode image to download.

This function locates the image specified by the *imageId* parameter from the specified microcode image library which is pointed to by the *imageLibrary* parameter. It then stops and resets the NPE, loads the firmware image onto the NPE, and then restarts the NPE.

This is a facility for users who wish to use an image from an external library of NPE firmware images. To use a standard image from the built-in library, see **ixNpeDINpeInitAndStart** instead.

**Note:**

A list of valid image IDs is included in this header file. See #defines with prefix **IX\_NPEDL\_NPEIMAGE\_...**

This function, along with **ixNpeDINpeInitAndStart** and **ixNpeDILoadedImageFunctionalityGet**, supercedes the following functions which are deprecated and will be removed completely in a

future release:

- ◇ **ixNpeDIImageDownload**
- ◇ **ixNpeDIAvailableImagesCountGet**
- ◇ **ixNpeDIAvailableImagesListGet**
- ◇ **ixNpeDILatestImageGet**
- ◇ **ixNpeDILoadedImageGet**
- ◇ **ixNpeDIMicrocodeImageLibraryOverride**
- ◇ **ixNpeDINpeExecutionStop**
- ◇ **ixNpeDINpeStopAndReset**
- ◇ **ixNpeDINpeExecutionStart**

**Precondition:**

- ◇ The Client is responsible for ensuring mutual access to the NPE.
- ◇ The image library supplied must be in the correct format for use by the NPE Downloader (IxNpeDI) component. Details of the library format are contained in the Functional Specification document for IxNpeDI.

**Postcondition:**

- ◇ The NPE Instruction Pipeline will be cleared if State Information has been downloaded.
- ◇ If the download fails with a critical error, the NPE may be left in an usable state.

**Returns:**

- ◇ IX\_SUCCESS if the download was successful;
- ◇ IX\_NPEDL\_PARAM\_ERR if a parameter error occurred
- ◇ IX\_NPEDL\_CRITICAL\_NPE\_ERR if a critical NPE error occurred during download
- ◇ IX\_NPEDL\_CRITICAL\_MICROCODE\_ERR if a critical microcode error occurred during download
- ◇ IX\_FAIL if NPE is not available or image is failed to be located. A warning is issued if the NPE is not present.

```
PUBLIC IX_STATUS ixNpeDIImageDownload ( IxNpeDIImageId * imageIdPtr,  
                                         BOOL          verify  
                                         )
```

Stop, reset, download microcode and finally start NPE.

**Parameters:**

- IxNpeDIImageId*\* [in] *imageIdPtr* – Pointer to Id of the microcode image to download.
- BOOL* [in] *verify* – ON/OFF option to verify the download. If ON (*verify* == TRUE), the Downloader will read back each word written to the NPE registers to ensure the download operation was successful.

Using the *imageIdPtr*, this function locates a particular image of microcode in the microcode image library in memory, and downloads the microcode to a particular NPE.

**Warning:**

**THIS FUNCTION HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDINpeInitAndStart** and

## **ixNpeDlCustomImageNpeInitAndStart.**

### **Precondition:**

- ◇ The Client is responsible for ensuring mutual access to the NPE.

- The Client should use **ixNpeDlLatestImageGet()** to obtain the latest version of the image before attempting download.

### **Postcondition:**

- The NPE Instruction Pipeline will be cleared if State Information has been downloaded.
- If the download fails with a critical error, the NPE may be left in an usable state.

### **Returns:**

- IX\_SUCCESS if the download was successful;
- IX\_NPEDL\_PARAM\_ERR if a parameter error occurred
- IX\_NPEDL\_CRITICAL\_NPE\_ERR if a critical NPE error occurred during download
- IX\_PARAM\_CRITICAL\_MICROCODE\_ERR if a critical microcode error occurred during download
- IX\_FAIL if NPE is not available or image is failed to be located. A warning is issued if the NPE is not present.

```
PUBLIC IX_STATUS ixNpeDlLatestImageGet ( IxNpeDlNpeId          npeId,  
                                         IxNpeDlFunctionalityId functionalityId,  
                                         IxNpeDlImageId *       imageIdPtr  
                                         )
```

This instructs NPE Downloader to get Id of the latest version for an image that is specified by the client.

### **Parameters:**

*IxNpeDlNpeId* [in] *npeId* – Id of the target NPE.  
*IxNpeDlFunctionalityId* [in] *functionalityId* – functionality of the image  
*IxNpeDlImageId \** [out] *imageIdPtr* – Pointer to the where the image id should be stored.

This function sets NPE Downloader to return the id of the latest version for image. The user will select the image by providing a particular NPE (specifying *npeId*) with particular functionality (specifying *FunctionalityId*). The most recent version available as determined by the highest Major and Minor revision numbers is returned in *imageIdPtr*.

### **Warning:**

**THIS FUNCTION HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDlNpeInitAndStart** and **ixNpeDlCustomImageNpeInitAndStart**.

### **Returns:**

- ◇ IX\_SUCCESS if the operation was successful
- ◇ IX\_NPEDL\_PARAM\_ERR if a parameter error occurred
- ◇ IX\_FAIL otherwise

```
PUBLIC IX_STATUS ixNpeDILoadedImageFunctionalityGet ( IxNpeDINpeId npeId,
                                                    UINT8 * functionalityId
                                                    )
```

Gets the functionality of the image last loaded on a particular NPE.

**Parameters:**

*IxNpeDINpeId* [in] *npeId* – Id of the target NPE.

UINT8\* [out] *functionalityId* – the functionality ID of the image last loaded on the NPE.

This function retrieves the functionality ID of the image most recently downloaded successfully to the specified NPE. If the NPE does not contain a valid image, this function returns a FAIL status.

**Warning:**

This function is not intended for general use, as a knowledge of how to interpret the functionality ID is required. As such, this function should only be used by other Access Layer components of the IXP400 Software Release.

**Precondition:**

**Postcondition:**

**Returns:**

- ◇ IX\_SUCCESS if the operation was successful
- ◇ IX\_NPEDL\_PARAM\_ERR if a parameter error occurred
- ◇ IX\_FAIL if the NPE does not have a valid image loaded

```
PUBLIC IX_STATUS ixNpeDILoadedImageGet ( IxNpeDINpeId npeId,
                                         IxNpeDIImageId * imageIdPtr
                                         )
```

Gets the Id of the image currently loaded on a particular NPE.

**Parameters:**

*IxNpeDINpeId* [in] *npeId* – Id of the target NPE.

*IxNpeDIImageId*\* [out] *imageIdPtr* – Pointer to the where the image id should be stored.

If an image of microcode was previously downloaded successfully to the NPE by NPE Downloader, this function returns in *imageIdPtr* the image Id of that image loaded on the NPE.

**Warning:**

**THIS FUNCTION HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release.

**Precondition:**

- ◇ The Client has allocated memory to the *imageIdPtr* pointer.

**Postcondition:**

**Returns:**

- ◇ IX\_SUCCESS if the operation was successful
- ◇ IX\_NPEDL\_PARAM\_ERR if a parameter error occurred
- ◇ IX\_FAIL if the NPE doesn't currently have a image loaded

```
IX_STATUS ixNpeDIMicrocodeImageLibraryOverride ( UINT32 * clientImageLibrary )
```

This instructs NPE Downloader to use client-supplied microcode image library.

**Parameters:**

UINT32\* [in] *clientImageLibrary* – Pointer to the client-supplied NPE microcode image library

This function sets NPE Downloader to use a client-supplied microcode image library instead of the standard image library which is included by the NPE Downloader. **This function is provided mainly for increased testability and should not be used in normal circumstances.** When not used, NPE Downloader will use a "built-in" image library, local to this component, which should always contain the latest microcode for the NPEs.

**Warning:**

**THIS FUNCTION HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDICustomImageNpeInitAndStart**.

**Precondition:**

- ◇ *clientImageLibrary* should point to a microcode image library valid for use by the NPE Downloader component.

**Postcondition:**

- ◇ the client-supplied image library will be used for all subsequent operations performed by the NPE Downloader

**Returns:**

- ◇ IX\_SUCCESS if the operation was successful
- ◇ IX\_NPEDL\_PARAM\_ERR if a parameter error occurred
- ◇ IX\_FAIL if the client-supplied image library did not contain a valid signature

```
PUBLIC IX_STATUS ixNpeDINpeExecutionStart ( IxNpeDINpeId npeId )
```

Starts code execution on a NPE.

**Parameters:**

*IxNpeDINpeId* [in] *npeId* – Id of the target NPE

Starts execution of code on a particular NPE. A client would typically use this after a download to NPE is performed, to start/restart code execution on the NPE.

**Note:**

It is no longer necessary to call this function after downloading a new image to the NPE. It is left on the API only to allow greater control of NPE execution if required. Where appropriate, use

**ixNpeDINpeInitAndStart** or **ixNpeDICustomImageNpeInitAndStart** instead.

**Warning:**

**THIS FUNCTION HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDINpeInitAndStart** and **ixNpeDICustomImageNpeInitAndStart**.

**Precondition:**

- ◇ The Client is responsible for ensuring mutual access to the NPE.
- ◇ Note that this function does not set the NPE Next Program Counter (NextPC), so it should be set beforehand if required by downloading appropriate State Information (using **ixNpeDIVersionDownload()**).

**Postcondition:**

**Returns:**

- ◇ **IX\_SUCCESS** if the operation was successful
- ◇ **IX\_NPEDL\_PARAM\_ERR** if a parameter error occurred
- ◇ **IX\_FAIL** otherwise

```
PUBLIC IX_STATUS ixNpeDINpeExecutionStop ( IxNpeDINpeId npeId )
```

Stops code execution on a NPE.

**Parameters:**

*IxNpeDINpeId* [in] *npeId* – Id of the target NPE

Stops execution of code on a particular NPE. This would typically be used by a client before a download to NPE is performed, to stop code execution on an NPE, unless **ixNpeDINpeStopAndReset()** is used instead. Unlike **ixNpeDINpeStopAndReset()**, this function only halts the NPE and leaves all registers and settings intact. This is useful, for example, between stages of a multi-stage download, to stop the NPE prior to downloading the next image while leaving the current state of the NPE intact..

**Warning:**

**THIS FUNCTION HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDINpeInitAndStart** and **ixNpeDICustomImageNpeInitAndStart**.

**Precondition:**

- ◇ The Client is responsible for ensuring mutual access to the NPE.

**Postcondition:**

**Returns:**

- ◇ **IX\_SUCCESS** if the operation was successful
- ◇ **IX\_NPEDL\_PARAM\_ERR** if a parameter error occurred
- ◇ **IX\_FAIL** otherwise



```
PUBLIC IX_STATUS ixNpeDINpeInitAndStart ( UINT32 npelImageId )
```

Stop, reset, download microcode (firmware) and finally start NPE.

**Parameters:**

UINT32 [in] *imageId* – Id of the microcode image to download.

This function locates the image specified by the *imageId* parameter from the default microcode image library which is included internally by this component. It then stops and resets the NPE, loads the firmware image onto the NPE, and then restarts the NPE.

**Note:**

A list of valid image IDs is included in this header file. See #defines with prefix IX\_NPEDL\_NPEIMAGE\_...

This function, along with **ixNpeDICustomImageNpeInitAndStart** and **ixNpeDILoadedImageFunctionalityGet**, supersedes the following functions which are deprecated and will be removed completely in a future release:

- ◇ **ixNpeDIImageDownload**
- ◇ **ixNpeDIAvailableImagesCountGet**
- ◇ **ixNpeDIAvailableImagesListGet**
- ◇ **ixNpeDILatestImageGet**
- ◇ **ixNpeDILoadedImageGet**
- ◇ **ixNpeDIMicrocodeImageLibraryOverride**
- ◇ **ixNpeDINpeExecutionStop**
- ◇ **ixNpeDINpeStopAndReset**
- ◇ **ixNpeDINpeExecutionStart**

**Precondition:**

- ◇ The Client is responsible for ensuring mutual access to the NPE.

**Postcondition:**

- ◇ The NPE Instruction Pipeline will be cleared if State Information has been downloaded.
- ◇ If the download fails with a critical error, the NPE may be left in an unusable state.

**Returns:**

- ◇ IX\_SUCCESS if the download was successful;
- ◇ IX\_NPEDL\_PARAM\_ERR if a parameter error occurred
- ◇ IX\_NPEDL\_CRITICAL\_NPE\_ERR if a critical NPE error occurred during download
- ◇ IX\_NPEDL\_CRITICAL\_MICROCODE\_ERR if a critical microcode error occurred during download
- ◇ IX\_FAIL if NPE is not available or image is failed to be located. A warning is issued if the NPE is not present.

```
PUBLIC IX_STATUS ixNpeDINpeStopAndReset ( IxNpeDINpeId npelId )
```

Stops and Resets an NPE.

**Parameters:**

*IxNpeDI**NpeId* [in] *npeId* – Id of the target NPE.

This function performs a soft NPE reset by writing reset values to particular NPE registers. Note that this does not reset NPE co-processors. This implicitly stops NPE code execution before resetting the NPE.

**Note:**

It is no longer necessary to call this function before downloading a new image to the NPE. It is left on the API only to allow greater control of NPE execution if required. Where appropriate, use **ixNpeDINpeInitAndStart** or **ixNpeDICustomImageNpeInitAndStart** instead.

**Warning:**

**THIS FUNCTION HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDINpeInitAndStart** and **ixNpeDICustomImageNpeInitAndStart**.

**Precondition:**

◇ The Client is responsible for ensuring mutual access to the NPE.

**Postcondition:**

**Returns:**

- ◇ IX\_SUCCESS if the operation was successful
- ◇ IX\_NPEDL\_PARAM\_ERR if a parameter error occurred
- ◇ IX\_FAIL otherwise

```
PUBLIC void ixNpeDIStatsReset ( void )
```

This function will reset the statistics of the IxNpeDI component.

**Returns:**

none

```
PUBLIC void ixNpeDIStatsShow ( void )
```

This function will display run-time statistics from the IxNpeDI component.

**Returns:**

none

```
PUBLIC IX_STATUS ixNpeDIUnload ( void )
```

This function will uninitialise the IxNpeDI component.

This function will uninitialise the IxNpeDI component. It should only be called once, and only if the IxNpeDI component has already been initialised by calling any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**
- **ixNpeDlImageDownload** (deprecated)
- **ixNpeDlNpeStopAndReset** (deprecated)
- **ixNpeDlNpeExecutionStop** (deprecated)
- **ixNpeDlNpeExecutionStart** (deprecated)

If possible, this function should be called before a soft reboot or unloading a kernel module to perform any clean up operations required for IxNpeDl.

The following actions will be performed by this function:

- Unmapping of any kernel memory mapped by IxNpeDl

**Returns:**

- ◊ IX\_SUCCESS if the operation was successful
- ◊ IX\_FAIL otherwise

# IXP425 NPE Image ID Definition

## [IXP425 NPE–Downloader (IxNpeDI) API]

Definition of NPE Image ID to be passed to **ixNpeDIInitAndStart()** as input of type **UINT32** which has the following fields format:.

### Defines

**#define IX\_NPEDL\_NPEIMAGE\_FIELD\_MASK**

*Mask for NPE Image ID's Field.*

**#define IX\_NPEDL\_NPEIMAGE\_BIT\_LOC\_NPEID**

*Location of NPE ID field in term of bit.*

**#define IX\_NPEDL\_NPEIMAGE\_BIT\_LOC\_FUNCTIONALITYID**

*Location of Functionality ID field in term of bit.*

**#define IX\_NPEDL\_NPEIMAGE\_BIT\_LOC\_MAJOR**

*Location of Major Release Number field in term of bit.*

**#define IX\_NPEDL\_NPEIMAGE\_BIT\_LOC\_MINOR**

*Location of Minor Release Number field in term of bit.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEA\_HSS0**

*NPE Image Id for NPE–A with HSS–0 Only feature. It supports 32 channelized and 4 packetized.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEA\_HSS0\_ATM\_SPHY\_1\_PORT**

*NPE Image Id for NPE–A with HSS–0 and ATM feature. For HSS, it supports 16/32 channelized and 4/0 packetized. For ATM, it supports AAL5, AAL0 and OAM for UTOPIA SPHY, 1 logical port, 32 VCs. It also has Fast Path support.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEA\_HSS0\_ATM\_MPHY\_1\_PORT**

*NPE Image Id for NPE–A with HSS–0 and ATM feature. For HSS, it supports 16/32 channelized and 4/0 packetized. For ATM, it supports AAL5, AAL0 and OAM for UTOPIA MPHY, 1 logical port, 32 VCs. It also has Fast Path support.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEA\_ATM\_MPHY\_12\_PORT**

*NPE Image Id for NPE–A with ATM–Only feature. It supports AAL5, AAL0 and OAM for UTOPIA MPHY, 12 logical ports, 32 VCs. It also has Fast Path support.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEA\_HSS\_2\_PORT**

*NPE Image Id for NPE–A with HSS–0 and HSS–1 feature. Each HSS port supports 32 channelized and 4 packetized.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEA\_DMA**

*NPE Image Id for NPE–A with DMA–Only feature.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEA\_WEP**

*NPE Image Id for NPE–A with ARC4 and WEP CRC engines.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEB\_ETH**  
*NPE Image Id for NPE–B with Ethernet–Only feature.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEB\_ETH\_FPATH**  
*NPE Image Id for NPE–B with Ethernet and Fast Path feature.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEB\_DMA**  
*NPE Image Id for NPE–B with DMA–Only feature.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEC\_ETH**  
*NPE Image Id for NPE–C with Eth–Only feature.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEC\_CRYPT0**  
*NPE Image Id for NPE–C with Crypto–Only feature. For Crypto, it supports DES, SHA–1, MD5.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEC\_CRYPT0\_AES**  
*NPE Image Id for NPE–C with Crypto–Only feature. For Crypto, it supports AES, DES, SHA–1, MD5.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEC\_CRYPT0\_ETH**  
*NPE Image Id for NPE–C with Crypto and Eth feature. For Crypto, it supports DES, SHA–1, MD5.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEC\_CRYPT0\_AES\_ETH**  
*NPE Image Id for NPE–C with Crypto and Eth feature. For Crypto, it supports AES, DES, SHA–1, MD5.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEC\_CRYPT0\_AES\_CCM**  
*NPE Image Id for NPE–C with Crypto–Only feature. For Crypto, it supports AES, CCM, DES, SHA–1, MD5.*

**#define IX\_NPEDL\_NPEIMAGE\_NPEC\_CRYPT0\_AES\_CCM\_ETH**  
*NPE Image Id for NPE–C with Crypto and Eth feature. For Crypto, it supports AES, CCM, DES, SHA–1, MD5. For Ethernet, MAC address learning disabled (but filtering is still enabled).*

**#define IX\_NPEDL\_NPEIMAGE\_NPEC\_DMA**  
*NPE Image Id for NPE–C with DMA–Only feature.*

---

## Detailed Description

Definition of NPE Image ID to be passed to **ixNpeDINpeInitAndStart()** as input of type **UINT32** which has the following fields format:.

Field [Bit Location]

-----  
NPE ID [31 – 24]

NPE Functionality ID [23 – 16]

## Define Documentation

```
#define IX_NPEDL_NPEIMAGE_BIT_LOC_FUNCTIONALITYID
```

Location of Functionality ID field in term of bit.

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **152** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_BIT_LOC_MAJOR
```

Location of Major Release Number field in term of bit.

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **163** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_BIT_LOC_MINOR
```

Location of Minor Release Number field in term of bit.

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **174** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_BIT_LOC_NPEID
```

Location of NPE ID field in term of bit.

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDlNpeInitAndStart** for more information.

Definition at line **141** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_FIELD_MASK
```

Mask for NPE Image ID's Field.

**Warning:**

**THIS #define HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDIInitAndStart** for more information.

Definition at line **130** of file **IxNpeDI.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEA_ATM_MPHY_12_PORT
```

NPE Image Id for NPE–A with ATM–Only feature. It supports AAL5, AAL0 and OAM for UTOPIA MPHY, 12 logical ports, 32 VCs. It also has Fast Path support.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDIInitAndStart**
- **ixNpeDICustomImageNpeInitAndStart**

Definition at line **231** of file **IxNpeDI.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEA_DMA
```

NPE Image Id for NPE–A with DMA–Only feature.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDIInitAndStart**
- **ixNpeDICustomImageNpeInitAndStart**

Definition at line **254** of file **IxNpeDI.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEA_HSS0
```

NPE Image Id for NPE–A with HSS–0 Only feature. It supports 32 channelized and 4 packetized.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDIInitAndStart**
- **ixNpeDICustomImageNpeInitAndStart**

Definition at line **190** of file **IxNpeDI.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEA_HSS0_ATM_MPHY_1_PORT
```

NPE Image Id for NPE–A with HSS–0 and ATM feature. For HSS, it supports 16/32 channelized and 4/0 packetized. For ATM, it supports AAL5, AAL0 and OAM for UTOPIA MPHY, 1 logical port, 32 VCs. It also has Fast Path support.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **218** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEA_HSS0_ATM_SPHY_1_PORT
```

NPE Image Id for NPE–A with HSS–0 and ATM feature. For HSS, it supports 16/32 channelized and 4/0 packetized. For ATM, it supports AAL5, AAL0 and OAM for UTOPIA SPHY, 1 logical port, 32 VCs. It also has Fast Path support.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **204** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEA_HSS_2_PORT
```

NPE Image Id for NPE–A with HSS–0 and HSS–1 feature. Each HSS port supports 32 channelized and 4 packetized.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **243** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEA_WEP
```

NPE Image Id for NPE–A with ARC4 and WEP CRC engines.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **265** of file **IxNpeDl.h**.



```
#define IX_NPEDL_NPEIMAGE_NPEB_DMA
```

NPE Image Id for NPE-B with DMA-Only feature.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **303** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEB_ETH
```

NPE Image Id for NPE-B with Ethernet-Only feature.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **281** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEB_ETH_FPATH
```

NPE Image Id for NPE-B with Ethernet and Fast Path feature.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **292** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTO
```

NPE Image Id for NPE-C with Crypto-Only feature. For Crypto, it supports DES, SHA-1, MD5.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **330** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTO_AES
```

NPE Image Id for NPE–C with Crypto–Only feature. For Crypto, it supports AES, DES, SHA–1, MD5.

**Note:**

This can only be used with the B0 silicon revision of the IXP425

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **344** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTOAES_CCM
```

NPE Image Id for NPE–C with Crypto–Only feature. For Crypto, it supports AES, CCM, DES, SHA–1, MD5.

**Note:**

This can only be used with the B0 silicon revision of the IXP425

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **384** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTOAES_CCM_ETH
```

NPE Image Id for NPE–C with Crypto and Eth feature. For Crypto, it supports AES, CCM, DES, SHA–1, MD5. For Ethernet, MAC address learning disabled (but filtering is still enabled).

**Note:**

This can only be used with the B0 silicon revision of the IXP425

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **400** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTOAES_ETH
```

NPE Image Id for NPE–C with Crypto and Eth feature. For Crypto, it supports AES, DES, SHA–1, MD5.

**Note:**

This can only be used with the B0 silicon revision of the IXP425

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **370** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEC_CRYPTOTH
```

NPE Image Id for NPE–C with Crypto and Eth feature. For Crypto, it supports DES, SHA–1, MD5.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **356** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEC_DMA
```

NPE Image Id for NPE–C with DMA–Only feature.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **411** of file **IxNpeDl.h**.

```
#define IX_NPEDL_NPEIMAGE_NPEC_ETH
```

NPE Image Id for NPE–C with Eth–Only feature.

This is intended for use as a parameter with any of the following functions:

- **ixNpeDlNpeInitAndStart**
- **ixNpeDlCustomImageNpeInitAndStart**

Definition at line **318** of file **IxNpeDl.h**.

# IXP425 NPE Message Handler (IxNpeMh) API

The public API for the IXP425 NPE Message Handler component.

## Data Structures

struct **IxNpeMhMessage**

*The 2-word message structure to send to and receive from the NPEs.*

## Defines

#define **IX\_NPEMH\_MIN\_MESSAGE\_ID**

*minimum valid message ID*

#define **IX\_NPEMH\_MAX\_MESSAGE\_ID**

*maximum valid message ID*

#define **IX\_NPEMH\_SEND\_RETRIES\_DEFAULT**

*default msg send retries*

## Typedefs

typedef

UINT32 **IxNpeMhMessageId**

*message ID*

typedef

void(\* **IxNpeMhCallback**)(IxNpeMhNpeId, IxNpeMhMessage)

*This prototype shows the format of a message callback function.*

## Enumerations

enum **IxNpeMhNpeId** {

**IX\_NPEMH\_NPEID\_NPEA,**

**IX\_NPEMH\_NPEID\_NPEB,**

**IX\_NPEMH\_NPEID\_NPEC,**

**IX\_NPEMH\_NUM\_NPES**

}

*The ID of a particular NPE.*

enum **IxNpeMhNpeInterrupts** {

**IX\_NPEMH\_NPEINTERRUPTS\_NO,**

**IX\_NPEMH\_NPEINTERRUPTS\_YES**

}

*Indicator specifying whether or not NPE interrupts should drive receiving of messages from the NPEs.*

## Functions

**IX\_STATUS ixNpeMhInitialize (IxNpeMhNpeInterrupts npeInterrupts)**

*This function will initialise the IxNpeMh component.*

**IX\_STATUS ixNpeMhUnload (void)**

*This function will uninitialise the IxNpeMh component.*

**IX\_STATUS ixNpeMhUnsolicitedCallbackRegister (IxNpeMhNpeId npeId, IxNpeMhMessageId messageId, IxNpeMhCallback unsolicitedCallback)**

*This function will register an unsolicited callback for a particular NPE and message ID.*

**IX\_STATUS ixNpeMhUnsolicitedCallbackForRangeRegister (IxNpeMhNpeId npeId, IxNpeMhMessageId minMessageId, IxNpeMhMessageId maxMessageId, IxNpeMhCallback unsolicitedCallback)**

*This function will register an unsolicited callback for a particular NPE and range of message IDs.*

**IX\_STATUS ixNpeMhMessageSend (IxNpeMhNpeId npeId, IxNpeMhMessage message, UINT32 maxSendRetries)**

*This function will send a message to a particular NPE.*

**IX\_STATUS ixNpeMhMessageWithResponseSend (IxNpeMhNpeId npeId, IxNpeMhMessage message, IxNpeMhMessageId solicitedMessageId, IxNpeMhCallback solicitedCallback, UINT32 maxSendRetries)**

*This function is equivalent to the **ixNpeMhMessageSend()** function, but must be used when the message being sent will solicited a response.*

**IX\_STATUS ixNpeMhMessagesReceive (IxNpeMhNpeId npeId)**

*This function will receive messages from a particular NPE and pass each message to the client via a solicited callback (for solicited messages) or an unsolicited callback (for unsolicited messages).*

**IX\_STATUS ixNpeMhShow (IxNpeMhNpeId npeId)**

*This function will display the current state of the IxNpeMh component.*

**IX\_STATUS ixNpeMhShowReset (IxNpeMhNpeId npeId)**

*This function will reset the current state of the IxNpeMh component.*

---

## Detailed Description

The public API for the IXP425 NPE Message Handler component.

---

## Define Documentation

```
#define IX_NPEMH_MAX_MESSAGE_ID
```

maximum valid message ID

Definition at line **68** of file **IxNpeMh.h**.

```
#define IX_NPEMH_MIN_MESSAGE_ID
```

minimum valid message ID

Definition at line **67** of file **IxNpeMh.h**.

```
#define IX_NPEMH_SEND_RETRIES_DEFAULT
```

default msg send retries

Definition at line **70** of file **IxNpeMh.h**.

---

## Typedef Documentation

```
IxNpeMhCallback
```

This prototype shows the format of a message callback function.

This prototype shows the format of a message callback function. The message callback will be passed the message to be handled and will also be told from which NPE the message was received. The message callback will either be registered by **ixNpeMhUnsolicitedCallbackRegister()** or passed as a parameter to **ixNpeMhMessageWithResponseSend()**. It will be called from within an ISR triggered by the NPE's "outFIFO not empty" interrupt (see **ixNpeMhInitialize()**). The parameters passed are the ID of the NPE that the message was received from, and the message to be handled.

**Re-entrancy:** This function is only a prototype, and will be implemented by the client. It does not need to be re-entrant.

Definition at line **134** of file **IxNpeMh.h**.

```
typedef UINT32 IxNpeMhMessageId
```

message ID

Definition at line **115** of file **IxNpeMh.h**.

---

## Enumeration Type Documentation

enum IxNpeMhNpeId

The ID of a particular NPE.

**Note:**

- In this context, for IXP425 Silicon (B0):
- ◊ NPE–A has HDLC, HSS, AAL and UTOPIA Coprocessors.
  - ◊ NPE–B has Ethernet Coprocessor.
  - ◊ NPE–C has Ethernet, AES, DES and HASH Coprocessors.
  - ◊ IXP425 Product Line have different combinations of coprocessors.

**Enumeration values:**

- IX\_NPEMH\_NPEID\_NPEA* ID for NPE–A.  
*IX\_NPEMH\_NPEID\_NPEB* ID for NPE–B.  
*IX\_NPEMH\_NPEID\_NPEC* ID for NPE–C.  
*IX\_NPEMH\_NUM\_NPES* Number of NPEs.

Definition at line **83** of file **IxNpeMh.h**.

enum IxNpeMhNpeInterrupts

Indicator specifying whether or not NPE interrupts should drive receiving of messages from the NPEs.

**Enumeration values:**

- IX\_NPEMH\_NPEINTERRUPTS\_NO* Don't use NPE interrupts.  
*IX\_NPEMH\_NPEINTERRUPTS\_YES* Do use NPE interrupts.

Definition at line **98** of file **IxNpeMh.h**.

---

## Function Documentation

IX\_STATUS ixNpeMhInitialize ( **IxNpeMhNpeInterrupts** npeInterrupts )

This function will initialise the IxNpeMh component.

**Parameters:**

- IxNpeMhNpeInterrupts* npeInterrupts (in) – This parameter dictates whether or not the IxNpeMh component will service NPE "outFIFO not empty" interrupts to trigger receiving and processing of messages from the NPEs. If not then the client

must use **ixNpeMhMessagesReceive()** to control message receiving and processing.

This function will initialise the IxNpeMh component. It should only be called once, prior to using the IxNpeMh component. The following actions will be performed by this function:

1. Initialization of internal data structures (e.g. solicited and unsolicited callback tables).
2. Configuration of the interface with the NPEs (e.g. enabling of NPE "outFIFO not empty" interrupts).
3. Registration of ISRs that will receive and handle messages when the NPEs' "outFIFO not empty" interrupts fire (if npeInterrupts equals IX\_NPEMH\_NPEINTERRUPTS\_YES).

**Returns:**

The function returns a status indicating success or failure.

```
IX_STATUS ixNpeMhMessageSend ( IxNpeMhNpeId    npeId,  
                                IxNpeMhMessage message,  
                                UINT32            maxSendRetries  
                                )
```

This function will send a message to a particular NPE.

**Parameters:**

*IxNpeMhNpeId* npeId (in) – The ID of the NPE to send the message to.  
*IxNpeMhMessage* message (in) – The message to send.  
*UINT32* maxSendRetries (in) – Max num. of retries to perform if the NPE's inFIFO is full.

This function will send a message to a particular NPE. It will be the client's responsibility to ensure that the message is properly formed. The return status will signify to the client if the message was successfully sent or not.

If the message is sent to the NPE then this function will return a status of success. Note that this will only mean the message has been placed in the NPE's inFIFO. There will be no way of knowing that the NPE has actually read the message, but once in the incoming message queue it will be safe to assume that the NPE will process it.

The inFIFO may fill up sometimes if the Xscale is sending messages faster than the NPE can handle them. This forces us to retry attempts to send the message until the NPE services the inFIFO. The client should specify a ceiling value for the number of retries suitable to their needs.

IX\_NPEMH\_SEND\_RETRIES\_DEFAULT can be used as a default value for the *maxSendRetries* parameter for this function. Each retry exceeding this default number will incur a blocking delay of 1 microsecond, to avoid consuming too much AHB bus bandwidth while performing retries.

Note this function **must** only be used for messages. that do not solicit responses. If the message being sent will solicit a response then the **ixNpeMhMessageWithResponseSend()** function **must** be used to ensure that the response is correctly handled.

**Re-entrancy:** This function will be callable from any thread at any time. IxOsServices will be used for any



necessary resource protection.

**Returns:**

The function returns a status indicating success or failure.

```
IX_STATUS ixNpeMhMessagesReceive ( IxNpeMhNpeId npeId )
```

This function will receive messages from a particular NPE and pass each message to the client via a solicited callback (for solicited messages) or an unsolicited callback (for unsolicited messages).

**Parameters:**

*IxNpeMhNpeId* *npeId* (in) – The ID of the NPE to receive and process messages from.

This function will receive messages from a particular NPE and pass each message to the client via a solicited callback (for solicited messages) or an unsolicited callback (for unsolicited messages).

If the *IxNpeMh* component is initialised to service NPE "outFIFO not empty" interrupts (see **ixNpeMhInitialize()**) then there is no need to call this function. This function is only provided as an alternative mechanism to control the receiving and processing of messages from the NPEs.

Note this function cannot be called from within an ISR as it will use resource protection mechanisms.

**Re-entrancy:** This function will be callable from any thread at any time. *IxOsServices* will be used for any necessary resource protection.

**Returns:**

The function returns a status indicating success or failure.

```
IX_STATUS ixNpeMhMessageWithResponseSend ( IxNpeMhNpeId      npeId,  
                                             IxNpeMhMessage    message,  
                                             IxNpeMhMessageId solicitedMessageId,  
                                             IxNpeMhCallback  solicitedCallback,  
                                             UINT32           maxSendRetries  
                                             )
```

This function is equivalent to the **ixNpeMhMessageSend()** function, but must be used when the message being sent will solicit a response.

**Parameters:**

*IxNpeMhNpeId* *npeId* (in) – The ID of the NPE to send the message to.

*IxNpeMhMessage* *message* (in) – The message to send.

*IxNpeMhMessageId* *solicitedMessageId* (in) – The ID of the solicited response message.

*IxNpeMhCallback* *solicitedCallback* (in) – The function to use to pass the response message back to the client. A value of NULL will cause the response message to be discarded.

*UINT32* *maxSendRetries* (in) – Max num. of retries to perform if the NPE's inFIFO is full.

This function is equivalent to the **ixNpeMhMessageSend()** function, but must be used when the message being sent will solicit a response.

The client must specify the ID of the solicited response message to allow the response to be recognised when it is received. The client must also specify a callback function to handle the received response. The IxNpeMh component will not offer the facility to send a message to a NPE and receive a response within the same context.

Note if the client is not interested in the response, specifying a NULL callback will cause the response message to be discarded.

The solicited callback will be stored and called some time later from an ISR that will be triggered by the NPE's "outFIFO not empty" interrupt (see **ixNpeMhInitialize()**) to handle the response message corresponding to the message sent. Response messages will be handled in the order they are received.

The inFIFO may fill up sometimes if the Xscale is sending messages faster than the NPE can handle them. This forces us to retry attempts to send the message until the NPE services the inFIFO. The client should specify a ceiling value for the number of retries suitable to their needs.

IX\_NPEMH\_SEND\_RETRIES\_DEFAULT can be used as a default value for the *maxSendRetries* parameter for this function. Each retry exceeding this default number will incur a blocking delay of 1 microsecond, to avoid consuming too much AHB bus bandwidth while performing retries.

**Re-entrancy:** This function will be callable from any thread at any time. IxOsServices will be used for any necessary resource protection.

**Returns:**

The function returns a status indicating success or failure.

```
IX_STATUS ixNpeMhShow ( IxNpeMhNpeId npeId )
```

This function will display the current state of the IxNpeMh component.

**Re-entrancy:** This function will be callable from any thread at any time. However, no resource protection will be used so as not to impact system performance. As this function is only reading statistical information then this is acceptable.

**Parameters:**

*IxNpeMhNpeId* npeId (in) – The ID of the NPE to display state information for.

**Returns:**

The function returns a status indicating success or failure.

```
IX_STATUS ixNpeMhShowReset ( IxNpeMhNpeId npeId )
```

This function will reset the current state of the IxNpeMh component.

**Re-entrancy:** This function will be callable from any thread at any time. However, no resource protection will be used so as not to impact system performance. As this function is only writing statistical information

then this is acceptable.

**Parameters:**

*IxNpeMhNpeId* npeId (in) – The ID of the NPE to reset state information for.

**Returns:**

The function returns a status indicating success or failure.

```
IX_STATUS ixNpeMhUnload ( void )
```

This function will uninitialise the IxNpeMh component.

This function will uninitialise the IxNpeMh component. It should only be called once, and only if the IxNpeMh component has already been initialised. No other IxNpeMh API functions should be called until **ixNpeMhInitialize** is called again. If possible, this function should be called before a soft reboot or unloading a kernel module to perform any clean up operations required for IxNpeMh.

The following actions will be performed by this function:

1. Unmapping of kernel memory mapped by the function **ixNpeMhInitialize**.

**Returns:**

The function returns a status indicating success or failure.

```
IX_STATUS ixNpeMhUnsolicitedCallbackForRangeRegister ( IxNpeMhNpeId      npeId,  
                                                       IxNpeMhMessageId minMessageId,  
                                                       IxNpeMhMessageId maxMessageId,  
                                                       IxNpeMhCallback   unsolicitedCallback  
                                                       )
```

This function will register an unsolicited callback for a particular NPE and range of message IDs.

**Parameters:**

*IxNpeMhNpeId* npeId (in) – The ID of the NPE whose messages the unsolicited callback will handle.

*IxNpeMhMessageId* minMessageId (in) – The minimum message ID in the range of message IDs the unsolicited callback will handle.

*IxNpeMhMessageId* maxMessageId (in) – The maximum message ID in the range of message IDs the unsolicited callback will handle.

*IxNpeMhCallback* unsolicitedCallback (in) – The unsolicited callback function. A value of NULL will deregister any previously registered callback(s) for this NPE and range of message IDs.

This function will register an unsolicited callback for a particular NPE and range of message IDs. It is a convenience function that is effectively the same as calling **ixNpeMhUnsolicitedCallbackRegister()** for each ID in the specified range. See **ixNpeMhUnsolicitedCallbackRegister()** for more information.

**Re-entrancy:** This function will be callable from any thread at any time. IxOsServices will be used for any necessary resource protection.

**Returns:**

The function returns a status indicating success or failure.

```
IX_STATUS ixNpeMhUnsolicitedCallbackRegister ( IxNpeMhNpeId      npeId,  
                                              IxNpeMhMessageId messageId,  
                                              IxNpeMhCallback   unsolicitedCallback  
                                              )
```

This function will register an unsolicited callback for a particular NPE and message ID.

**Parameters:**

*IxNpeMhNpeId* *npeId* (in) – The ID of the NPE whose messages the unsolicited callback will handle.  
*IxNpeMhMessageId* *messageId* (in) – The ID of the messages the unsolicited callback will handle.  
*IxNpeMhCallback* *unsolicitedCallback* (in) – The unsolicited callback function. A value of NULL will deregister any previously registered callback for this NPE and message ID.

This function will register an unsolicited message callback for a particular NPE and message ID.

If an unsolicited callback is already registered for the specified NPE and message ID then the callback will be overwritten. Only one client will be responsible for handling a particular message ID associated with a NPE. Registering a NULL unsolicited callback will deregister any previously registered callback.

The callback function will be called from an ISR that will be triggered by the NPE's "outFIFO not empty" interrupt (see **ixNpeMhInitialize()**) to handle any unsolicited messages of the specific message ID received from the NPE. Unsolicited messages will be handled in the order they are received.

If no unsolicited callback can be found for a received message then it is assumed that the message is solicited.

If more than one client may be interested in a particular unsolicited message then the suggested strategy is to register a callback for the message that can itself distribute the message to multiple clients as necessary.

See also **ixNpeMhUnsolicitedCallbackForRangeRegister()**.

**Re-entrancy:** This function will be callable from any thread at any time. IxOsServices will be used for any necessary resource protection.

**Returns:**

The function returns a status indicating success or failure.

# IXP425 OS Memory Buffer Management (IxOsBuffMgt) API

IXP425 OS Memory Buffer Management.

## Data Structures

```
struct __IX_MBUF
```

## Defines

```
#define m_net_pool
#define m_next
#define m_len
#define m_data
#define m_type
#define m_flags
#define m_nextpkt
#define m_act
#define m_pkthdr
#define SIZEOF_IP_HEADER
#define IX_MBUF_MDATA(m_blk_ptr)
    Return pointer to the data in the mbuf.

#define IX_MBUF_MLEN(m_blk_ptr)
    Return the data length.

#define IX_MBUF_TYPE(m_blk_ptr)
    Return the data type in the mbuf.

#define IX_MBUF_NEXT_BUFFER_IN_PKT_PTR(m_blk_ptr)
    Return pointer to the next mbuf in a single packet.

#define IX_MBUF_NEXT_PKT_IN_CHAIN_PTR(m_blk_ptr)
    Return pointer to the next packet in the chain.

#define IX_MBUF_ALLOCATED_BUFF_LEN(m_blk_ptr)
    Return the allocated buffer size.

#define IX_MBUF_PKT_LEN(m_blk_ptr)
    Return the total length of all the data in the mbuf chain for this packet.

#define IX_MBUF_FLAGS(m_blk_ptr)
    Return the buffer flags.

#define IX_MBUF_NET_POOL(m_blk_ptr)
```

*Return pointer to a network pool.*

## Typedefs

typedef \_\_IX\_MBUF **IX\_MBUF**

---

## Detailed Description

IXP425 OS Memory Buffer Management.

---

## Define Documentation

```
#define IX_MBUF_ALLOCATED_BUFF_LEN ( m_blk_ptr )
```

Return the allocated buffer size.

Definition at line **162** of file **IxOsBuffMgt.h**.

```
#define IX_MBUF_FLAGS ( m_blk_ptr )
```

Return the buffer flags.

Definition at line **176** of file **IxOsBuffMgt.h**.

```
#define IX_MBUF_MDATA ( m_blk_ptr )
```

Return pointer to the data in the mbuf.

Definition at line **127** of file **IxOsBuffMgt.h**.

```
#define IX_MBUF_MLEN ( m_blk_ptr )
```

Return the data length.

Definition at line **134** of file **IxOsBuffMgt.h**.

```
#define IX_MBUF_NET_POOL ( m_blk_ptr )
```

Return pointer to a network pool.

Definition at line **184** of file **IxOsBuffMgt.h**.

```
#define IX_MBUF_NEXT_BUFFER_IN_PKT_PTR ( m_blk_ptr )
```

Return pointer to the next mbuf in a single packet.

Definition at line **148** of file **IxOsBuffMgt.h**.

```
#define IX_MBUF_NEXT_PKT_IN_CHAIN_PTR ( m_blk_ptr )
```

Return pointer to the next packet in the chain.

Definition at line **155** of file **IxOsBuffMgt.h**.

```
#define IX_MBUF_PKT_LEN ( m_blk_ptr )
```

Return the total length of all the data in the mbuf chain for this packet.

Definition at line **169** of file **IxOsBuffMgt.h**.

```
#define IX_MBUF_TYPE ( m_blk_ptr )
```

Return the data type in the mbuf.

Definition at line **141** of file **IxOsBuffMgt.h**.

# IXP425 OS Memory Buffer Pool Management (IxOsBuffPoolMgt) API

The Public API for the Buffer Pool Management component.

## Data Structures

struct **IxMbufPool**

*Implementation of buffer pool structure for use with non-VxWorks OS.*

## Defines

#define **IX\_MBUF\_POOL\_SIZE\_ALIGN**(size)

*This macro takes an integer as an argument and rounds it up to be a multiple of the memory cache-line size.*

#define **IX\_MBUF\_POOL\_MBUF\_AREA\_SIZE\_ALIGNED**(count)

*This macro calculates, from the number of mbufs required, the size of the memory area required to contain the mbuf headers for the buffers in the pool. The size to be used for each mbuf header is rounded up to a multiple of the cache-line size, to ensure each mbuf header aligns on a cache-line boundary. This macro is used by **IX\_MBUF\_POOL\_MBUF\_AREA\_ALLOC**().*

#define **IX\_MBUF\_POOL\_DATA\_AREA\_SIZE\_ALIGNED**(count, size)

*This macro calculates, from the number of mbufs required and the size of the data portion for each mbuf, the size of the data memory area required. The size is adjusted to ensure alignment on cache line boundaries. This macro is used by **IX\_MBUF\_POOL\_DATA\_AREA\_ALLOC**().*

#define **IX\_MBUF\_POOL\_MBUF\_AREA\_ALLOC**(count, memAreaSize)

*Allocates the memory area needed for the number of mbuf headers specified by count. This macro ensures the mbuf headers align on cache line boundaries. This macro evaluates to a pointer to the memory allocated.*

#define **IX\_MBUF\_POOL\_DATA\_AREA\_ALLOC**(count, size, memAreaSize)

*Allocates the memory pool for the data portion of the pool mbufs. The number of mbufs is specified by count. The size of the data portion of each mbuf is specified by size. This macro ensures the mbufs are aligned on cache line boundaries. This macro evaluates to a pointer to the memory allocated.*

#define **IX\_MBUF\_POOL\_FREE\_COUNT**(poolPtr)

*Returns the number of free buffers currently in the specified pool.*

#define **IX\_MBUF\_MAX\_POOLS**

*The maximum number of pools that can be allocated.*



```

#define IX_MBUF_POOL_NAME_LEN
    The maximum string length of the pool name.

#define IX_MBUF_POOL_INIT(poolPtrPtr, count, size, name)
    Wrapper macro for ixOsBuffPoolInit() See function description below for details.

#define IX_MBUF_POOL_INIT_NO_ALLOC(poolPtrPtr, bufPtr, dataPtr, count, size, name)
    Wrapper macro for ixOsBuffPoolInitNoAlloc() See function description below for details.

#define IX_MBUF_POOL_GET(poolPtr, bufPtrPtr)
    Wrapper macro for ixOsBuffPoolUnchainedBufGet() See function description below for details.

#define IX_MBUF_POOL_PUT(bufPtr)
    Wrapper macro for ixOsBuffPoolBufFree() See function description below for details.

#define IX_MBUF_POOL_PUT_CHAIN(bufPtr)
    Wrapper macro for ixOsBuffPoolBufChainFree() See function description below for details.

#define IX_MBUF_POOL_SHOW(poolPtr)
    Wrapper macro for ixOsBuffPoolShow() See function description below for details.

#define IX_MBUF_POOL_MDATA_RESET(bufPtr)
    Wrapper macro for ixOsBuffPoolBufDataPtrReset() See function description below for details.

```

## Typedefs

```

typedef
IxMbufPool IX_MBUF_POOL
    The buffer pool structure, mapped to appropriate OS-specific implementation.

```

## Enumerations

```

enum IxMbufPoolAllocationType {
    IX_MBUF_POOL_TYPE_SYS_ALLOC,
    IX_MBUF_POOL_TYPE_USER_ALLOC
}
    Used to indicate how the pool memory was allocated.

```

## Functions

```

IX_STATUS ixOsBuffPoolInit (IX_MBUF_POOL **poolPtrPtr, int count, int size, char *name)

```

*This function creates a new buffer pool.*

## **IX\_MBUF\_POOL**

**\* ixOsBuffPoolAllocate** (void)

*This function allocates buffers from the available pool.*

UINT32 **ixOsBuffPoolDataAreaSizeGet** (int count, int size)

*This function calculates the size of data memory required to create a new buffer pool.*

IX\_MBUF \* **ixOsBuffPoolMbufInit** (int mbufSizeAligned, int dataSizeAligned, **IX\_MBUF\_POOL** \*poolPtr)

*Allocate memory for mbuf and data and initialise mbuf header fields.*

UINT32 **ixOsBuffPoolMbufAreaSizeGet** (int count)

*This function calculates the size of mbuf memory required to create a new buffer pool.*

IX\_STATUS **ixOsBuffPoolInitNoAlloc** (**IX\_MBUF\_POOL** \*\*poolPtrPtr, void \*poolBufPtr, void \*poolDataPtr, int count, int size, char \*name)

*This function creates a new buffer pool, with user-allocated memory.*

IX\_STATUS **ixOsBuffPoolUnchainedBufGet** (**IX\_MBUF\_POOL** \*poolPtr, IX\_MBUF \*\*newBufPtrPtr)

*This function gets a buffer from the buffer pool.*

IX\_MBUF \* **ixOsBuffPoolBufFree** (IX\_MBUF \*bufPtr)

*This function returns a buffer to the buffer pool.*

void **ixOsBuffPoolBufChainFree** (IX\_MBUF \*bufPtr)

*This function returns a buffer chain to the buffer pool.*

IX\_STATUS **ixOsBuffPoolShow** (**IX\_MBUF\_POOL** \*poolPtr)

*This function prints pool statistics.*

IX\_STATUS **ixOsBuffPoolBufDataPtrReset** (IX\_MBUF \*bufPtr)

*This function resets the data pointer of a buffer.*

IX\_STATUS **ixOsBuffPoolUninit** (**IX\_MBUF\_POOL** \*pool)

*Unitalize buffer pool.*

---

## **Detailed Description**

The Public API for the Buffer Pool Management component.

---

## **Define Documentation**

```
#define IX_MBUF_MAX_POOLS
```

The maximum number of pools that can be allocated.

**Note:**

This can safely be increased if more pools are required.

Definition at line **241** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_DATA_AREA_ALLOC ( count,  
                                     size,  
                                     memAreaSize )
```

Allocates the memory pool for the data portion of the pool mbufs. The number of mbufs is specified by *count*. The size of the data portion of each mbuf is specified by *size*. This macro ensures the mbufs are aligned on cache line boundaries. This macro evaluates to a pointer to the memory allocated.

**Parameters:**

*int* [in] *count* – the number of mbufs the pool will contain

*int* [in] *size* – the desired size (in bytes) required for the data portion of each mbuf. Note that this size may be rounded up to ensure alignment on cache–line boundaries.

*int* [out] *memAreaSize* – the total amount of memory allocated

**Returns:**

void \* – a pointer to the allocated memory area

Definition at line **200** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_DATA_AREA_SIZE_ALIGNED ( count,  
                                             size )
```

This macro calculates, from the number of mbufs required and the size of the data portion for each mbuf, the size of the data memory area required. The size is adjusted to ensure alignment on cache line boundaries. This macro is used by **IX\_MBUF\_POOL\_DATA\_AREA\_ALLOC()**.

**Note:**

Refer to the WindRiver "VxWorks 5.5 OS Libraries API Reference" manual for "netBufLib" library documentation, which explains the vxWorks implementation of this macro below.

**Parameters:**

*int* [in] *count* – The number of mbufs in the pool.

*int* [in] *size* – The desired size for each mbuf data portion. This size will be rounded up to a multiple of the cache–line size to ensure alignment on cache–line boundaries for each data block.

**Returns:**

int – the total size required for the pool data area (aligned)

Definition at line **161** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_FREE_COUNT ( poolPtr )
```

Returns the number of free buffers currently in the specified pool.

**Parameters:**

*IX\_MBUF\_POOL* \* [in] poolPtr – a pointer to the pool to query

**Returns:**

int – the number of free buffers in the pool

Definition at line **214** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_GET ( poolPtr,  
                           bufPtrPtr )
```

Wrapper macro for **ixOsBuffPoolUnchainedBufGet()** See function description below for details.

Definition at line **319** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_INIT ( poolPtrPtr,  
                           count,  
                           size,  
                           name      )
```

Wrapper macro for **ixOsBuffPoolInit()** See function description below for details.

Definition at line **301** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_INIT_NO_ALLOC ( poolPtrPtr,  
                                     bufPtr,  
                                     dataPtr,  
                                     count,  
                                     size,  
                                     name      )
```

Wrapper macro for **ixOsBuffPoolInitNoAlloc()** See function description below for details.

Definition at line **310** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_MBUF_AREA_ALLOC ( count,  
                                       memAreaSize )
```

Allocates the memory area needed for the number of mbuf headers specified by *count*. This macro ensures the mbuf headers align on cache line boundaries. This macro evaluates to a pointer to the memory allocated.

**Parameters:**

*int* [in] count – the number of mbufs the pool will contain  
*int* [out] memAreaSize – the total amount of memory allocated

**Returns:**

void \* – a pointer to the allocated memory area

Definition at line **178** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_MBUF_AREA_SIZE_ALIGNED ( count )
```

This macro calculates, from the number of mbufs required, the size of the memory area required to contain the mbuf headers for the buffers in the pool. The size to be used for each mbuf header is rounded up to a multiple of the cache-line size, to ensure each mbuf header aligns on a cache-line boundary. This macro is used by **IX\_MBUF\_POOL\_MBUF\_AREA\_ALLOC()**.

**Note:**

Refer to the WindRiver "VxWorks 5.5 OS Libraries API Reference" manual for "netBufLib" library documentation, which explains the vxWorks implementation of this macro below.

**Parameters:**

*int* [in] count – the number of buffers the pool will contain

**Returns:**

int – the total size required for the pool mbuf area (aligned)

Definition at line **137** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_MDATA_RESET ( bufPtr )
```

Wrapper macro for **ixOsBuffPoolBufDataPtrReset()** See function description below for details.

Definition at line **355** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_NAME_LEN
```

The maximum string length of the pool name.

Definition at line **248** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_PUT ( bufPtr )
```

Wrapper macro for **ixOsBuffPoolBufFree()** See function description below for details.

Definition at line **328** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_PUT_CHAIN ( bufPtr )
```

Wrapper macro for **ixOsBuffPoolBufChainFree()** See function description below for details.

Definition at line **337** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_SHOW ( poolPtr )
```

Wrapper macro for **ixOsBuffPoolShow()** See function description below for details.

Definition at line **346** of file **IxOsBuffPoolMgt.h**.

```
#define IX_MBUF_POOL_SIZE_ALIGN ( size )
```

This macro takes an integer as an argument and rounds it up to be a multiple of the memory cache–line size.

***Parameters:***

*int* [in] size – the size integer to be rounded up

***Returns:***

int – the size, rounded up to a multiple of the cache–line size

Definition at line **113** of file **IxOsBuffPoolMgt.h**.

---

## Typedef Documentation

```
typedef NET_POOL IX_MBUF_POOL
```

The buffer pool structure, mapped to appropriate OS–specific implementation.

Definition at line **282** of file **IxOsBuffPoolMgt.h**.

---

## Enumeration Type Documentation

```
enum IxMbufPoolAllocationType
```

Used to indicate how the pool memory was allocated.

***Enumeration values:***

<i>IX_MBUF_POOL_TYPE_SYS_ALLOC</i>	mbuf pool allocated by the system
<i>IX_MBUF_POOL_TYPE_USER_ALLOC</i>	mbuf pool allocated by the user

## Function Documentation

**IX\_MBUF\_POOL** \* ixOsBuffPoolAllocate ( void )

This function allocates buffers from the available pool.

Thread Safe: yes

**Returns:**

◇ IX\_MBUF\_POOL – pointer to mbuf pool.

ixOsBuffPoolBufChainFree ( IX\_MBUF \* bufPtr )

This function returns a buffer chain to the buffer pool.

**Parameters:**

IX\_MBUF \* [in] bufPtr – Pointer to head of the chain..

This function returns a buffer chain to the pool, making the buffers available again to **ixOsBuffPoolUnchainedBufGet()**. The buffer pointed to by *bufPtr* can be chained or unchained. If it is chained, all buffers in the chain will be returned to the pool.

Thread Safe: yes

**Precondition:**

◇ *bufPtr* should point to a valid IX\_MBUF structure

**Postcondition:**

◇ The buffer (or chain of buffers) supplied will be returned to the pool for reuse.

**Returns:**

◇ none

ixOsBuffPoolBufDataPtrReset ( IX\_MBUF \* bufPtr )

This function resets the data pointer of a buffer.

**Parameters:**

IX\_MBUF \* [in] bufPtr – Pointer to a valid IX\_MBUF buffer.

This function resets the data pointer of a buffer to point to the start of the memory area allocated to the buffer for data (the buffer payload).

**Note:**

WARNING – This function can NOT be used if BOTH of following conditions are true:

- ◊ the pool was created using **ixOsBuffPoolInitNoAlloc()**
- ◊ a NULL value was supplied at the time of creation for the *bufDataPtr* parameter of **ixOsBuffPoolInitNoAlloc()**. See the API description of the function **ixOsBuffPoolInitNoAlloc()**.

Thread Safe: yes

**Precondition:**

- ◊ *bufPtr* should point to a valid IX\_MBUF structure
- ◊ A non-NULL pointer to the data memory area was supplied when the pool was created (see note above)
- ◊ The data pointer is pointing to somewhere within the buffer payload

**Postcondition:**

- ◊ The data pointer of the mbuf header will point to the start of the data payload section of the buffer, as it did when it was originally obtained from the pool

**Returns:**

- ◊ IX\_SUCCESS if the operation was successful
- ◊ IX\_FAIL if the operation was not successful

```
ixOsBuffPoolBufFree ( IX_MBUF * bufPtr )
```

This function returns a buffer to the buffer pool.

**Parameters:**

*IX\_MBUF* \* [in] *bufPtr* – Pointer to a valid IX\_MBUF buffer.

This function returns a buffer to the pool, making it available again to **ixOsBuffPoolUnchainedBufGet()**. The buffer pointed to by *bufPtr* can be chained or unchained. If it is chained, only the head of the chain will be freed to the pool, and a pointer to the next buffer in the chain will be returned to the caller.

Thread Safe: yes

**Precondition:**

- ◊ *bufPtr* should point to a valid IX\_MBUF structure

**Postcondition:**

- ◊ The buffer supplied will be returned to the pool for reuse.

**Returns:**

- ◊ If supplied buffer was chained, a pointer to the next buffer in the chain is returned
- ◊ Otherwise NULL is returned

```
ixOsBuffPoolDataAreaSizeGet ( int count,  
                             int size
```



)

This function calculates the size of data memory required to create a new buffer pool.

**Parameters:**

*int* [in] *count* – The number of buffers to have in the pool.

*int* [in] *size* – The size of each buffer in the pool.

Thread Safe: yes

**Returns:**

◇ UINT32 the memory size required

```
ixOsBuffPoolInit ( IX_MBUF_POOL ** poolPtrPtr,  
                  int count,  
                  int size,  
                  char * name  
                  )
```

This function creates a new buffer pool.

**Parameters:**

**IX\_MBUF\_POOL** \*\* [out] *poolPtrPtr* – Pointer to a pool pointer.

*int* [in] *count* – The number of buffers to have in the pool.

*int* [in] *size* – The size of each buffer in the pool.

*char* \* [in] *name* – A name string for the pool (used in pool show).

This function initialises a pool of *count* buffers, each of size *size*. It allocates memory for the pool, fills in the pool and buffer data structures, and returns a pointer to the pool in the *poolPtrPtr* parameter. This pointer should be used with other functions on this API to use the pool. In the current implementation, only a limited number of pools can be allocated. The number of pools is decided by the value of **IX\_MBUF\_MAX\_POOLS**.

**Note:**

This function has 2 implementations, depending on which OS the code is compiled for. If compiled for VxWorks, an mbuf pool will be created using the VxWorks "netBufLib" OS library. This will produce a pool of mbufs which can be used with the netBufLib library routines if required. If compiled for a different OS, a pool of generic buffers will be produced. These may need to be converted to a different buffer format (such as sk\_bufs for Linux) to be used with OS network buffer manipulation routines if required. See the header file **IxOsBuffMgt.h** which maps the buffer implementations for each OS supported.

Thread Safe: no

**Precondition:**

◇ *poolPtrPtr* should point to a valid **IX\_MBUF\_POOL** pointer

**Postcondition:**

- ◇ A pool will be initialised and all memory required by the pool will be dynamically allocated from memory.

**Returns:**

- ◇ IX\_SUCCESS if the operation was successful
- ◇ IX\_FAIL if the pool could not be created

```
ixOsBuffPoolInitNoAlloc ( IX_MBUF_POOL ** poolPtrPtr,
                        void * poolBufPtr,
                        void * poolDataPtr,
                        int count,
                        int size,
                        char * name
                      )
```

This function creates a new buffer pool, with user-allocated memory.

**Parameters:**

<i>IX_MBUF_POOL</i> **	[out] poolPtrPtr – Pointer to a pool pointer.
<i>void</i>	* [in] poolBufPtr – pointer to memory allocated with <b>IX_MBUF_POOL_MBUF_AREA_ALLOC()</b>
<i>void</i>	* [in] poolDataPtr – pointer to memory allocated with <b>IX_MBUF_POOL_DATA_AREA_ALLOC()</b>
<i>int</i>	[in] count – The number of buffers to have in the pool.
<i>int</i>	[in] size – The size of each buffer in the pool (i.e. the amount of payload data octets each buffer can hold).
<i>char</i>	* [in] name – A name string for the pool (used in pool show).

This function initialises a pool of *count* buffers, each of size *size*. It fills in the pool and buffer data structures, and returns a pointer to the pool in the *poolPtrPtr* parameter. This pointer should be used with other functions on this API to use the pool. In the current implementation, only a limited number of pools can be allocated. The number of pools is decided by the value of *IX\_MBUF\_MAX\_POOLS*.

**Note:**

This function has 2 implementations, depending on which OS the code is compiled for. If compiled for VxWorks, an mbuf pool will be created using the VxWorks "netBufLib" OS library. This will produce a pool of mbufs which can be used with the netBufLib library routines if required. If compiled for a different OS, a pool of generic buffers will be produced. These may need to be converted to a different buffer format (such as sk\_buffs for Linux) to be used with OS network buffer manipulation routines if required. See the header file **IxOsBuffMgt.h** which maps the buffer implementations for each OS supported.

The pointer to the data area can optionally be NULL, to indicate that the data memory area for the mbuf payload will be assigned by the user later on. In this case, it is expected that the user would assign the data pointer of each mbuf returned by **ixOsBuffPoolUnchainedBufGet()**. This also means that the function **ixOsBuffPoolBufDataPtrReset()** cannot be used on buffers from this pool.  
**WARNING** – This pointer CANNOT be NULL if VxWorks implementation is used!

Thread Safe: no

**Precondition:**

- ◇ *poolPtrPtr* should point to a valid IX\_MBUF\_POOL pointer
- ◇ The memory required for the pool, for mbuf structures and data (if required), should be allocated with the specified macros. See params *poolBufPtr* and *poolDataPtr*

**Postcondition:**

- ◇ A pool will be initialised and all memory required by the pool will be dynamically allocated from memory.

**Returns:**

- ◇ IX\_SUCCESS if the operation was successful
- ◇ IX\_FAIL if the pool could not be created

```
ixOsBuffPoolMbufAreaSizeGet ( int count )
```

This function calculates the size of mbuf memory required to create a new buffer pool.

**Parameters:**

*int* [in] *count* – The number of buffers to have in the pool.

Thread Safe: yes

**Returns:**

- ◇ UINT32 the memory size required

```
ixOsBuffPoolMbufInit ( int mbufSizeAligned,
                       int dataSizeAligned,
                       IX_MBUF_POOL * poolPtr
                       )
```

Allocate memory for mbuf and data and initialise mbuf header fields.

This function allocates memory for an individual mbuf contained in an mbuf wrapper, defined above. The purpose of the wrapper is to add extra fields to the mbuf header which are hidden from the user. These extra fields are intended for use only by IxOsBuffPoolMgt.

**Parameters:**

*int* [in] *mbufSizeAligned* – Size aligned mbuf.  
*int* [in] *dataSizeAligned* – Size aligned data.  
 IX\_MBUF\_POOL \*[in]*poolPtr* – Pointer to buffer pool.

**Returns:**

IX\_MBUF\_POOL \*[in]*poolPtr* – Pointer to MBuf

```
ixOsBuffPoolShow ( IX_MBUF_POOL * poolPtr )
```

This function prints pool statistics.

**Parameters:**

*IX\_MBUF\_POOL* \* [in] *poolPtr* – A pointer to a valid pool.

This function prints pool statistics, such as the number of free buffers in each pool. The actual statistics printed depends on the implementation which may differ between platforms. This function can serve as a useful debugging aid.

Thread Safe: yes

**Precondition:**

◇ *poolPtr* should point to a valid *IX\_MBUF\_POOL* structure

**Postcondition:****Returns:**

- ◇ *IX\_SUCCESS* if the operation was successful
- ◇ *IX\_FAIL* if the pool statistics could not be printed.

```
ixOsBuffPoolUnchainedBufGet ( IX_MBUF_POOL * poolPtr,
                             IX_MBUF ** newBufPtrPtr
                             )
```

This function gets a buffer from the buffer pool.

**Parameters:**

*IX\_MBUF\_POOL* \* [in] *poolPtr* – Pointer to a valid pool.

*IX\_MBUF* \*\* [out] *newBufPtrPtr* – A pointer to a valid *IX\_MBUF* pointer.

This function gets a free buffer from the specified pool, and returns a pointer to the buffer in the *newBufPtrPtr* parameter. The buffer obtained will be a single unchained buffer of the size specified when the pool was initialised.

Thread Safe: yes

**Precondition:**

◇ *poolPtr* should point to a valid *IX\_MBUF\_POOL* structure

◇ *newBufPtrPtr* should point to a valid *IX\_MBUF* pointer

**Postcondition:**

- ◇ A free buffer will be allocated from the pool and *newBufPtrPtr* can be dereferenced to access the pointer to the buffer.

**Returns:**

- ◇ *IX\_SUCCESS* if the operation was successful
- ◇ *IX\_FAIL* if a free buffer could not be obtained

```
ixOsBuffPoolUninit ( IX_MBUF_POOL * pool )
```

Unitialize buffer pool.

***Parameters:***

*IX\_MBUF\_POOL* \*[in]pool – pointer to  
buffer pool

***Returns:***

- ◇ IX\_SUCCESS if the operation was  
successful
- ◇ IX\_FAIL if the operation was not  
successful

# IXP425 OS Cache MMU (IxOsCacheMMU) API

This service provides services to the access components and codelets to abstract out any cache coherency issues and mmu mappings.

## Defines

```
#define IX_ACC_CACHE_ENABLED  
    This macro enable cached memory in Access layers.  
  
#define IX_ACC_DRV_DMA_MALLOC(size)  
    Allocate memory for driver use, that will be shared between XScale and NPE's.  
  
#define IX_ACC_DRV_DMA_FREE(ptr, size)  
    Free memory allocated from IX_ACC_DRV_DMA_MALLOC.  
  
#define IX_MMU_VIRTUAL_TO_PHYSICAL_TRANSLATION(addr)  
    Return a virtual address for the provided physical address.  
  
#define IX_MMU_PHYSICAL_TO_VIRTUAL_TRANSLATION(addr)  
    Return a physical address for the provided virtual address.  
  
#define IX_XSCALE_CACHE_LINE_SIZE  
    IX_XSCALE_CACHE_LINE_SIZE = size of cache line for both flush and invalidate.  
  
#define IX_ACC_DRAM_PHYS_OFFSET  
    PHYS_OFFSET = Physical DRAM offset..  
  
#define IX_ACC_DATA_CACHE_INVALIDATE(addr, size)  
    Invalidate a cache range.  
  
#define IX_ACC_DATA_CACHE_FLUSH(addr, size)  
    Flush a cache range to physical memory.
```

## Functions

```
void * ixOsServCacheDmaAlloc (UINT32 size)  
    Allocate memory for driver use, that will be shared between XScale and NPE's.  
  
void ixOsServCacheDmaFree (void *ptr, UINT32 size)  
    Free memory allocated from ixOsServCacheDmaAlloc.
```

---

## Detailed Description

This service provides services to the access components and codelets to abstract out any cache coherency issues and mmu mappings.

---

## Define Documentation

```
#define IX_ACC_CACHE_ENABLED
```

This macro enable cached memory in Access layers.

When defined, this macro enable the use of cached memory in the access layers.

To disable cache on mbufs, #undef the macro

---

Definition at line **89** of file **IxOsCacheMMU.h**.

```
#define IX_ACC_DATA_CACHE_FLUSH ( addr,  
                                size )
```

Flush a cache range to physical memory.

Flush a cache range to physical memory.

### **Note:**

This is typically done prior to submitting a buffer to the NPE's which you expect the NPE to read from. Entire Cache lines will be flushed.

- If memory space used is non cached, then this function does may be null.
- Functionality required:
  1. Non-Cached space : Flush CPU WB, No cache Flush.
  2. Write Through Enabled : Flush CPU WB, No cache Flush.
  3. Copy Back Enabled : Flush CPU WB, Invalidate area specified

### **Attention:**

There are different implementations for this macro

- Linux OS implementation:
  - ◆ #define **IX\_ACC\_DATA\_CACHE\_FLUSH(addr,size)** clean\_dcache\_range((\_\_u32)addr, (\_\_u32)addr + size )
- VxWorks OS implementation:
  - ◆ #define **IX\_ACC\_DATA\_CACHE\_FLUSH(addr,size)** do { cacheFlush(DATA\_CACHE, addr, size); cachePipeFlush(); } while(0)
- VxWorks OS implementation (write-through mode):
  - ◆ #define **IX\_ACC\_DATA\_CACHE\_FLUSH(addr,size)** cachePipeFlush();

- default implementation (uncached memory):  
 ♦ #define **IX\_ACC\_DATA\_CACHE\_FLUSH(addr,size)**

---

Definition at line **364** of file **IxOsCacheMMU.h**.

```
#define IX_ACC_DATA_CACHE_INVALIDATE ( addr,  
                                     size )
```

Invalidate a cache range.

Invalidate a cache range.

**Note:**

This is typically done prior to submitting a buffer to the NPE's which you expect the NPE to populate with data.

**Attention:**

The size argument must be a multiple of cacheline size, i.e. a multiple of 32bytes for the XSCALE. The argument shall be rounded up to the next 32byte boundry. Extreme care must be taken when invalidating cache lines due.

- If memory space used is non cached, then this function may be null.
- Functionality required:
  1. Non-Cached space : No functionality required.
  2. Write Through Enabled : Invalidate area specified
  3. Copy Back Enabled : Invalidate area specified

**Attention:**

There are different implementations for this macro

- Linux OS implementation:
  - ♦ #define **IX\_ACC\_DATA\_CACHE\_INVALIDATE(addr,size)**  
 invalidate\_dcache\_range((\_\_u32)addr, (\_\_u32)addr + size )
- VxWorks OS implementation:
  - ♦ #define **IX\_ACC\_DATA\_CACHE\_INVALIDATE(addr,size)**  
 cacheInvalidate(DATA\_CACHE, addr, size)
- default implementation:
  - ♦ #define **IX\_ACC\_DATA\_CACHE\_INVALIDATE(addr,size)**

---

Definition at line **363** of file **IxOsCacheMMU.h**.

```
#define IX_ACC_DRAM_PHYS_OFFSET
```

PHYS\_OFFSET = Physical DRAM offset..

**Attention:**



There are different implementations for this macro

- Linux OS implementation:
  - ◆ #define IX\_ACC\_DRAM\_PHYS\_OFFSET (PHYS\_OFFSET)
- VxWorks OS implementation:
  - ◆ #define IX\_ACC\_DRAM\_PHYS\_OFFSET (0x00000000UL)

---

Definition at line **352** of file **IxOsCacheMMU.h**.

```
#define IX_ACC_DRV_DMA_FREE ( ptr,  
                             size )
```

Free memory allocated from IX\_ACC\_DRV\_DMA\_MALLOC.

This function frees the memory allocated from *IX\_ACC\_DRV\_DMA\_MALLOC*.

**Parameters:**

*void* \* ptr – pointer to the memory area to be freed.  
*UINT32* size – number of bytes of memory allocated.

**Returns:**

*void*

**See also:**

**IX\_ACC\_DRV\_DMA\_MALLOC**

---

Definition at line **174** of file **IxOsCacheMMU.h**.

```
#define IX_ACC_DRV_DMA_MALLOC ( size )
```

Allocate memory for driver use, that will be shared between XScale and NPE's.

This macro is used allocate memory for driver use, that will be shared between XScale and NPE's.

**Note:**

The buffer allocated with have the system defined attributes, and as such the Invalidate and flush macros functionality must be updated.

The buffer allocated is aligned on a cache line boundary.

**Parameters:**

*UINT32* size – number of bytes of memory requested.

**Returns:**

*void* \* Pointer to memory that can be used between XScale and NPE's.

*See also:*

**IX\_ACC\_DRV\_DMA\_FREE**

---

Definition at line **155** of file **IxOsCacheMMU.h**.

```
#define IX_MMU_PHYSICAL_TO_VIRTUAL_TRANSLATION ( addr )
```

Return a physical address for the provided virtual address.

This macro return a physical address for the provided virtual address

**Attention:**

There are different implementations for this macro

- Linux OS implementation:
    - ◆ #define **IX\_MMU\_VIRTUAL\_TO\_PHYSICAL\_TRANSLATION(addr)** ((addr) ? virt\_to\_phys((void\*)(addr)) : 0)
  - VxWorks OS implementation:
    - ◆ #define **IX\_MMU\_PHYSICAL\_TO\_VIRTUAL\_TRANSLATION(addr)** (addr)
  - default implementation:
    - ◆ #define **IX\_MMU\_PHYSICAL\_TO\_VIRTUAL\_TRANSLATION(addr)** (addr)
- 

Definition at line **362** of file **IxOsCacheMMU.h**.

```
#define IX_MMU_VIRTUAL_TO_PHYSICAL_TRANSLATION ( addr )
```

Return a virtual address for the provided physical address.

This macro return a virtual address for the provided physical address.

**Attention:**

There are different implementations for this macro

- Linux OS implementation:
    - ◆ #define **IX\_MMU\_VIRTUAL\_TO\_PHYSICAL\_TRANSLATION(addr)** ((addr) ? virt\_to\_phys((void\*)(addr)) : 0)
  - VxWorks OS implementation:
    - ◆ #define **IX\_MMU\_VIRTUAL\_TO\_PHYSICAL\_TRANSLATION(addr)** (addr)
  - default implementation:
    - ◆ #define **IX\_MMU\_VIRTUAL\_TO\_PHYSICAL\_TRANSLATION(addr)** (addr)
- 

Definition at line **361** of file **IxOsCacheMMU.h**.

```
#define IX_XSCALE_CACHE_LINE_SIZE
```

IX\_XSCALE\_CACHE\_LINE\_SIZE = size of cache line for both flush and invalidate.

---

Definition at line 366 of file **IxOsCacheMMU.h**.

---

## Function Documentation

```
void* ixOsServCacheDmaAlloc ( UINT32 size )
```

Allocate memory for driver use, that will be shared between XScale and NPE's.

Allocate memory for driver use, that will be shared between XScale and NPE's.

**Note:**

The buffer allocated with have the system defined atributes, and as such the Invalidate and flush macros functionality must be updated.

The buffer allocated is aligned on a cache line boundary.

**Parameters:**

*UINT32 size* – number of bytes of memory requested.

**Returns:**

void \* Pointer to memory that can be used between XScale and NPE's.

**See also:**

**ixOsServCacheDmaFree**

---

```
void ixOsServCacheDmaFree ( void * ptr,  
                           UINT32 size  
                           )
```

Free memory allocated from ixOsServCacheDmaAlloc.

This function frees the memory allocated from *ixOsServCacheDmaAlloc*.

**Parameters:**

*void* \**ptr* – pointer to the memory area to be freed.

*UINT32 size* – number of bytes of memory allocated.

**Returns:**

void

**See also:**

**ixOsServCacheDmaMalloc**

---

# IXP425 OS Services (IxOsServices) API

This service provides a very thin layer of OS dependency services.

## Defines

```
#define IX_OSSERV_QMGR_MAP_SIZE  
    Queue Manager map size.  
  
#define IX_OSSERV_EXP_REG_MAP_SIZE  
    Exp Bus Registers map size.  
  
#define IX_OSSERV_UART1_MAP_SIZE  
    UART1 map size.  
  
#define IX_OSSERV_UART2_MAP_SIZE  
    UART2 map size.  
  
#define IX_OSSERV_PMU_MAP_SIZE  
    PMU map size.  
  
#define IX_OSSERV_OSTS_MAP_SIZE  
    OS Timers map size.  
  
#define IX_OSSERV_NPEA_MAP_SIZE  
    NPE A map size.  
  
#define IX_OSSERV_NPEB_MAP_SIZE  
    NPE B map size.  
  
#define IX_OSSERV_NPEC_MAP_SIZE  
    NPE C map size.  
  
#define IX_OSSERV_ETHA_MAP_SIZE  
    Eth A map size.  
  
#define IX_OSSERV_ETHB_MAP_SIZE  
    Eth B map size.  
  
#define IX_OSSERV_USB_MAP_SIZE  
    USB map size.  
  
#define IX_OSSERV_GPIO_MAP_SIZE  
    GPIO map size.  
  
#define IX_OSSERV_EXP_BUS_MAP_SIZE  
    Expansion bus map size.
```

```
#define IX_OSSERV_EXP_BUS_CS1_MAP_SIZE
    CS1 map size.

#define IX_OSSERV_EXP_BUS_CS4_MAP_SIZE
    CS4 map size.

#define IX_OSSERV_MEM_MAP(requestedPhysicalAddress, size)
    Maps an I/O memory zone.

#define IX_OSSERV_MEM_UNMAP(requestedVirtualAddress)
    unmaps a previously mapped I/O memory zone

#define IX_OSSERV_MMAP_VIRT_TO_PHYS_TRANSLATION(virtualAddress)
    provides a mapped memory–aware virtual to physical translation
```

## Typedefs

```
typedef int IX_IRQ_STATUS
    Defines flag to indicate IRQ status.

typedef SEM_ID IxMutex
    Mutex object.

typedef SEM_ID IxFastMutex
    Fast mutex object.
```

## Enumerations

```
enum IxOsServTraceLevels {
    LOG_NONE,
    LOG_USER,
    LOG_FATAL,
    LOG_ERROR,
    LOG_WARNING,
    LOG_MESSAGE,
    LOG_DEBUG1,
    LOG_DEBUG2,
    LOG_DEBUG3,
    LOG_ALL
}
    Trace levels.
```

## Functions

```
PUBLIC
IX_STATUS ixOsServIntBind (int level, void(*routine)(void *), void *parameter)
```

*binds a routine to a hardware interrupt*

PUBLIC  
IX\_STATUS **ixOsServIntUnbind** (int level)  
*unbinds a routine from a hardware interrupt*

PUBLIC int **ixOsServIntLock** (void)  
*locks out IRQs*

PUBLIC void **ixOsServIntUnlock** (int lockKey)  
*unlocks IRQs*

PUBLIC int **ixOsServIntLevelSet** (int level)  
*sets the interrupt level*

PUBLIC  
IX\_STATUS **ixOsServMutexInit** (IxMutex \*mutex)  
*initializes a mutex*

PUBLIC  
IX\_STATUS **ixOsServMutexLock** (IxMutex \*mutex)  
*locks the given mutex*

PUBLIC  
IX\_STATUS **ixOsServMutexUnlock** (IxMutex \*mutex)  
*unlocks the given mutex*

PUBLIC  
IX\_STATUS **ixOsServMutexDestroy** (IxMutex \*mutex)  
*destroys a mutex object*

PUBLIC  
IX\_STATUS **ixOsServFastMutexInit** (IxFastMutex \*mutex)  
*initializes a fast mutex*

PUBLIC  
IX\_STATUS **ixOsServFastMutexTryLock** (IxFastMutex \*mutex)  
*attempts to lock the fast mutex object*

PUBLIC  
IX\_STATUS **ixOsServFastMutexUnlock** (IxFastMutex \*mutex)  
*unlocks the fast mutex object*

PUBLIC int **ixOsServLog** (int level, char \*format, int arg1, int arg2, int arg3, int arg4, int arg5, int arg6)  
*logs a formatted message*

PUBLIC int **ixOsServLogLevelSet** (int level)  
*sets the logging level*

PUBLIC void **ixOsServSleep** (int microseconds)

*execution block for a number of microseconds*

PUBLIC void **ixOsServTaskSleep** (int milliseconds)  
*preemptive execution block for a number of milliseconds*

PUBLIC unsigned  
int **ixOsServTimestampGet** (void)  
*used to retrieve the system timestamp*

PUBLIC void **ixOsServUnload** (void)  
*Used to un-map memory.*

PUBLIC void **ixOsServYield** (void)  
*Yields execution of the current thread.*

---

## Detailed Description

This service provides a very thin layer of OS dependency services.

This file contains the API to the functions which are some what OS dependant and would require porting to a particular OS. A primary focus of the component development is to make them as OS independent as possible. All other components should abstract their OS dependency to this module. Services overview

1. Trace Service – a simple debugging mechanism, with compile time debug trace level (depends on existent OS logging feature e.g. in Linux use kprintf, in VxWorks logMsg)
2. Mutual Exclusion
  - ◆ interrupt binding and locking mechanisms
  - ◆ mutex locks and fast mutexes
3. Timer Services
  - ◆ timed delays, busy loop – microsecond granularity
  - ◆ timed delays, OS dependent yielding – millisecond granularity
  - ◆ timestamp measurements – XScale core clock granularity

## OsServices I/O Memory Allocation and Access Routines

List of OSes and operating modes supported: (Please update the list when adding a new OS or mode)

- VxWorks BE
- VxWorks LE
- Linux BE

## Usage Information:

**IxOsServicesMemAccess.h** defines OS/Endianess mode, memory mapped I/O access macros, NPE-shared memory routines, SDRAM coherency mode, default component coherency mode and static/dynamic memory mapping for every component. The symbols defined below can be used and some overridden in the component-specific section of **IxOsServicesComponents.h**.



### *OS/Endianness defines:*

One of the following symbols will be defined:

**IX\_OSSERV\_VXWORKS\_BE** – component is compiled for VxWorks Big Endian

**IX\_OSSERV\_VXWORKS\_LE** – component is compiled for VxWorks Little Endian

**IX\_OSSERV\_LINUX\_BE** – component is compiled for Linux Big Endian

### *SDRAM coherency mode:*

One of the following symbols will be defined:

**IX\_SDRAM\_BE** – SDRAM is in Big Endian mode (default for Big Endian builds)

**IX\_SDRAM\_LE\_ADDRESS\_COHERENT** – SDRAM is in Little Endian, Address Coherent Mode (not supported by current software)

**IX\_SDRAM\_LE\_DATA\_COHERENT** – SDRAM is in Little Endian, Data Coherent Mode (default for Little Endian builds)

### *Static/dynamic memory mapping:*

**IX\_STATIC\_MEMORY\_MAP** – component uses statically I/O mapped memory (default)

**IX\_DYNAMIC\_MEMORY\_MAP** – component uses OS-specific dynamically mapped I/O memory (define this in the component specific section of **IxOsServicesComponents.h** to override the previous default value)

### *Component coherency mode (define or override defaults in the component section of IxOsServicesComponents.h):*

**CSR\_BE\_MAPPING** – component uses I/O memory in Big Endian mode (default in Big Endian Builds)

**CSR\_LE\_ADDRESS\_COHERENT\_MAPPING** – component uses I/O memory in Little Endian, Address Coherent mode

**CSR\_LE\_DATA\_COHERENT\_MAPPING** – component uses I/O memory in Little Endian, Data Coherent mode

**CSR\_NO\_MAPPING** – component does not use I/O memory (I/O read/write macros are not available)

## **Macros for memory mapped I/O access:**

Unless **CSR\_NO\_MAPPING** is defined, each component will have access to the following set of macros for reading and writing word (32 bit), short (16 bit) and byte (8 bits) data. The macros will perform all the necessary endianness conversions and use appropriate read and write functions in OSes where this is required.

The addresses (wAddr, sAddr, bAddr) should be volatile 32-bit pointers to **UINT32**, **UINT16** and **UINT8** respectively. The data (wData, sData, bData) should be **UINT32**, **UINT16** and respectively **UINT8** values.

**IX\_OSSERV\_READ\_LONG(wAddr)** – returns the 32-bit value at address wAddr

**IX\_OSSERV\_READ\_SHORT(sAddr)** – returns the 16-bit value at address sAddr

**IX\_OSSERV\_READ\_BYTE(bAddr)** – returns the 8-bit value at address bAddr

**IX\_OSSERV\_WRITE\_LONG(wAddr, wData)** – writes the 32-bit wData at address wAddr

**IX\_OSSERV\_WRITE\_SHORT(sAddr, sData)** – writes the 16-bit sData at address sAddr

**IX\_OSSERV\_WRITE\_BYTE(bAddr, bData)** – writes the 8-bit bData at address bAddr

## Macros for sharing data with the NPEs:

Each component will have access to the following set of macros for reading and writing word (32 bit) and short (16 bit) data, plus a macro for copying within SDRAM an array of words to be shared with an NPE. The macros will perform all the necessary endianness conversions depending on the SDRAM endianness and coherency mode to guarantee correct sharing of data between XScale components and NPEs.

The addresses (wAddr, sAddr, wSrcAddr and wDestAddr) should be 32-bit pointers to UINT32 and UINT16 respectively (volatility is NOT required). The data (wData and sData) should be UINT32 and respectively UINT16 values. The word count for the copy macro (wCount) should be a UINT32 value.

**IX\_OSSERV\_READ\_NPE\_SHARED\_LONG(wAddr)** – returns the 32-bit value written by the NPE at address wAddr

**IX\_OSSERV\_READ\_NPE\_SHARED\_SHORT(sAddr)** – returns the 16-bit value written by the NPE at address sAddr

**IX\_OSSERV\_WRITE\_NPE\_SHARED\_LONG(wAddr, wData)** – writes 32-bit wData to be read by the NPE at address wAddr

**IX\_OSSERV\_WRITE\_NPE\_SHARED\_SHORT(sAddr, sData)** – writes 16-bit sData to be read by the NPE at address sAddr

**IX\_OSSERV\_COPY\_NPE\_SHARED\_LONG\_ARRAY(wDestAddr, wSrcAddr, wCount)** – copies wCount 32-bit words shared with an NPE from wSrcAddr to wDestAddr

**IX\_OSSERV\_SWAP\_NPE\_SHARED\_LONG(wData)** – returns the correctly converted (if necessary) 32-bit value between NPE and SDRAM representation

---

## Define Documentation

```
#define IX_OSSERV_ETHA_MAP_SIZE
```

Eth A map size.

Definition at line **71** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_ETHB_MAP_SIZE
```

Eth B map size.

Definition at line **72** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_EXP_BUS_CS1_MAP_SIZE
```

CS1 map size.

Definition at line **76** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_EXP_BUS_CS4_MAP_SIZE
```

CS4 map size.

Definition at line **77** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_EXP_BUS_MAP_SIZE
```

Expansion bus map size.

Definition at line **75** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_EXP_REG_MAP_SIZE
```

Exp Bus Registers map size.

Definition at line **63** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_GPIO_MAP_SIZE
```

GPIO map size.

Definition at line **74** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_MEM_MAP ( requestedPhysicalAddress,  
                             size )
```

Maps an I/O memory zone.

***Parameters:***

*requestedAddress* UINT32 (in) – physical address to map

*size* UINT32 (in) – size of map (should be large enough to hold the largest offset access made in this zone)

This macro maps an I/O mapped physical memory zone of the given size into a virtual memory zone accessible by the caller and returns a cookie – the start address of the virtual memory zone.

IX\_MMU\_PHYS\_TO\_VIRT\_TRANSLATION should NOT therefore be used on the returned virtual address. The memory zone should be unmapped using IX\_OSSERV\_MEM\_UNMAP once the caller has finished using this zone (e.g. on driver unload) using the cookie as parameter. The IX\_OSSERV\_READ/WRITE\_LONG/SHORT macros should be used to read and write the mapped memory, adding the necessary offsets to the address cookie.

***Warning:***

Not to be called from interrupt level

**Note:**

The size parameter is only used for identifying a suitable (i.e. large enough) map in the global memory map (ixOsServGlobalMemoryMap). It is NOT used to actually map the memory zone. Instead, the zone indicated in the global memory map is used. Mapping will work only if the zone is predefined in the global memory map.

**Returns:**

the mapped virtual address or NULL if the operation could not be completed. This virtual address (cookie) has to be saved and used to unmap the memory zone during any clean-up or unload operation, using the IX\_OSSERV\_MEM\_UNMAP macro.

Definition at line **215** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_MEM_UNMAP ( requestedVirtualAddress )
```

unmaps a previously mapped I/O memory zone

**Parameters:**

*requestedAddress* UINT32 (in) – cookie (virtual address) to unmap

This macro unmaps a previously mapped I/O memory zone using the cookie obtained in the mapping operation. The memory zone in question becomes unavailable to the caller once unmapped and the cookie should be discarded.

This macro cannot fail if the given parameter is correct and does not return a value.

**Warning:**

Not to be called from interrupt level

Definition at line **237** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_MMAP_VIRT_TO_PHYS_TRANSLATION ( virtualAddress )
```

provides a mapped memory-aware virtual to physical translation

**Parameters:**

*virtualAddress* UINT32 (in) – cookie (virtual address) to translate

This macro searches through the global memory mapped for a virtualAddress match; if found, it will return the physical address defined in the map. Otherwise it will default to IX\_MMU\_VIRTUAL\_TO\_PHYSICAL\_TRANSLATION.

**Returns:**

the physical address translation

Definition at line **253** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_NPEA_MAP_SIZE
```

NPE A map size.

Definition at line **68** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_NPEB_MAP_SIZE
```

NPE B map size.

Definition at line **69** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_NPEC_MAP_SIZE
```

NPE C map size.

Definition at line **70** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_OSTS_MAP_SIZE
```

OS Timers map size.

Definition at line **67** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_PMU_MAP_SIZE
```

PMU map size.

Definition at line **66** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_QMGR_MAP_SIZE
```

Queue Manager map size.

Definition at line **62** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_UART1_MAP_SIZE
```

UART1 map size.

Definition at line **64** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_UART2_MAP_SIZE
```

UART2 map size.

Definition at line **65** of file **IxOsServicesMemMap.h**.

```
#define IX_OSSERV_USB_MAP_SIZE
```

USB map size.

Definition at line **73** of file **IxOsServicesMemMap.h**.

---

## Typedef Documentation

```
typedef int IX_IRQ_STATUS
```

Defines flag to indicate IRQ status.

Definition at line **96** of file **IxOsServices.h**.

```
typedef pthread_mutex_t IxFastMutex
```

Fast mutex object.

Definition at line **115** of file **IxOsServices.h**.

```
typedef pthread_mutex_t IxMutex
```

Mutex object.

Definition at line **102** of file **IxOsServices.h**.

---

## Enumeration Type Documentation

```
enum IxOsServTraceLevels
```

Trace levels.

**Enumeration values:**

<i>LOG_NONE</i>	No trace level.
<i>LOG_USER</i>	Set trace level to user.
<i>LOG_FATAL</i>	Set trace level to fatal.
<i>LOG_ERROR</i>	Set trace level to error.
<i>LOG_WARNING</i>	Set trace level to warning.
<i>LOG_MESSAGE</i>	Set trace level to message.
<i>LOG_DEBUG1</i>	Set trace level to debug1.
<i>LOG_DEBUG2</i>	Set trace level to debug2.
<i>LOG_DEBUG3</i>	Set trace level to debug3.
<i>LOG_ALL</i>	Set trace level to all.

Definition at line 135 of file **IxOsServices.h**.

---

## Function Documentation

```
PUBLIC IX_STATUS ixOsServFastMutexInit ( IxFastMutex * mutex )
```

initializes a fast mutex

**Parameters:**

*mutex* **IxFastMutex** \* (in) – pointer to the mutex object

Initializes a fast mutex, placing it in "unlocked" state.

Can be called from interrupt level: yes

**Returns:**

IX\_SUCCESS if the operation succeeded or IX\_FAIL  
otherwise

```
PUBLIC IX_STATUS ixOsServFastMutexTryLock ( IxFastMutex * mutex )
```

attempts to lock the fast mutex object

**Parameters:**

*mutex* IxFastMutex \* (in) – pointer to the mutex object

If the mutex is in "unlocked" state it becomes locked and the function returns "IX\_SUCCESS". If the mutex is already locked the function returns immediately with a IX\_FAIL result.

Can be called from interrupt level: yes

**Returns:**

IX\_SUCCESS if the operation succeeded or IX\_FAIL otherwise

```
PUBLIC IX_STATUS ixOsServFastMutexUnlock ( IxFastMutex * mutex )
```

unlocks the fast mutex object

**Parameters:**

*mutex* IxFastMutex \* (in) – pointer to the mutex object

Unlocks the given mutex object if locked, otherwise returns an error.

Can be called from interrupt level: yes

**Returns:**

IX\_SUCCESS if the operation succeeded or IX\_FAIL otherwise

```
PUBLIC IX_STATUS ixOsServIntBind ( int level,  
                                void(* routine)(void *),  
                                void * parameter  
                                )
```

binds a routine to a hardware interrupt

**Parameters:**

*level* int (in) – interrupt level to bind to

*routine* void (\*)(void \*) (in) – routine to connect

*parameter* void \* (in) – parameter to pass to the routine when called

This functions binds the specified C routine to an interrupt level. When called, the "parameter" value will be passed to the routine.

Can be called from interrupt level: no

**Returns:**

IX\_SUCCESS if the operation succeeded or IX\_FAIL otherwise

```
PUBLIC int ixOsServIntLevelSet ( int level )
```



sets the interrupt level

**Parameters:**

*level* int (in) – new interrupt level

This routine changes the interrupt mask in the status register to take on the value specified by *level*. Interrupts are locked out at or, depending on the implementation, below that level.

Can be called from interrupt level: yes

**Returns:**

the previous interrupt level

**Warning:**

Do not call system functions when interrupts are locked

```
PUBLIC int ixOsServIntLock ( void )
```

locks out IRQs

This function disables IRQs, returning a lock key which can be used later with `ixOsServIntUnlock` to re-enable the IRQs.

Can be called from interrupt level: yes

**Returns:**

a lock key used by `ixOsServIntUnlock` to unlock IRQs

**Warning:**

Do not call system routines when IRQs are locked

```
PUBLIC IX_STATUS ixOsServIntUnbind ( int level )
```

unbinds a routine from a hardware interrupt

**Parameters:**

*vector* int (in) – interrupt level to unbind from

This function unbinds from an interrupt level a C routine previously bound with `ixOsServIntBind`

Can be called from interrupt level: no

**Returns:**

`IX_SUCCESS` if the operation succeeded or `IX_FAIL` otherwise

```
PUBLIC void ixOsServIntUnlock ( int lockKey )
```

unlocks IRQs

**Parameters:**

*lockKey* int (in) – lock key previously obtained with  
**ixOsServIntLock()**

This function reenables the IRQs locked out by **ixOsServIntLock()**.

Can be called from interrupt level: yes

```
PUBLIC int ixOsServLog ( int    level,
                        char * format,
                        int    arg1,
                        int    arg2,
                        int    arg3,
                        int    arg4,
                        int    arg5,
                        int    arg6
                      )
```

logs a formatted message

**Parameters:**

*level* int (in) – trace level  
*format* char \* (in) – format string, similar to printf()  
*arg1* int (in) – first argument to display  
*arg2* int (in) – second argument to display  
*arg3* int (in) – third argument to display  
*arg4* int (in) – fourth argument to display  
*arg5* int (in) – fifth argument to display  
*arg6* int (in) – sixth argument to display

This function logs the specified message via the logging task. It is similar to printf(), however it requires a log level and a fixed number of arguments. Unless called with the LOG\_USER log level it will prefix the user message with a tag string like "[debug2]".

Can be called from interrupt level: yes

**Returns:**

the number of characters written

```
PUBLIC int ixOsServLogLevelSet ( int level )
```

sets the logging level

**Parameters:**

*level* int (in) – logging level

This function sets the maximum level allowed for logging. Setting it to LOG\_WARNING will disable all levels like LOG\_MESSAGE, LOG\_DEBUG1 etc. It returns the previous level which can be useful for restoring log states.

Can be called from interrupt level: yes

**Returns:**

the previous logging level

```
PUBLIC IX_STATUS ixOsServMutexDestroy ( IxMutex * mutex )
```

destroys a mutex object

**Parameters:**

*mutex* IxMutex \* (in) – pointer to mutex to destroy

Can be called from interrupt level: no

Destroys the mutex object, freeing the resources it might hold.

```
PUBLIC IX_STATUS ixOsServMutexInit ( IxMutex * mutex )
```

initializes a mutex

**Parameters:**

*mutex* IxMutex \* (in) – pointer to mutex to initialize

Initializes the given mutex (MUTual EXclusion device)

Can be called from interrupt level: no

**Returns:**

IX\_SUCCESS if the operation succeeded or IX\_FAIL  
otherwise

```
PUBLIC IX_STATUS ixOsServMutexLock ( IxMutex * mutex )
```

locks the given mutex

**Parameters:**

*mutex* IxMutex \* (in) – pointer to mutex to lock

If the mutex is unlocked it becomes locked and owned by the caller, and the function returns immediately.  
If the mutex is already locked calling this function will suspend the caller until the mutex is unlocked.

Can be called from interrupt level: no

**Returns:**

IX\_SUCCESS if the operation succeeded or IX\_FAIL otherwise

**Warning:**

Two or more consecutive calls from the same thread of **ixOsServMutexLock()** are likely to deadlock the calling thread. Mutexes are NOT recursive.

```
PUBLIC IX_STATUS ixOsServMutexUnlock ( IxMutex * mutex )
```

unlocks the given mutex

**Parameters:**

*mutex* **IxMutex** \* (in) – pointer to mutex to unlock

Unlocks the given mutex, returning it to the "unlocked" state. If the mutex was not locked an error is returned.

Can be called from interrupt level: yes

**Returns:**

IX\_SUCCESS if the operation succeeded or IX\_FAIL otherwise

```
PUBLIC void ixOsServSleep ( int microseconds )
```

execution block for a number of microseconds

**Parameters:**

*microseconds* int (in) – delay to block execution for

This function blocks the calling task using a timed busy loop.

Can be called from interrupt level: yes, except for the first invocation

```
PUBLIC void ixOsServTaskSleep ( int milliseconds )
```

preemptive execution block for a number of milliseconds

**Parameters:**

*milliseconds* int (in) – delay to block execution for

This function blocks the calling task using an OS-dependent preemptive timed delay.

Can be called from interrupt level: no

```
PUBLIC int ixOsServTimestampGet ( void )
```

used to retrieve the system timestamp

Can be called from interrupt level: yes, except for the first invocation

**Returns:**

the current timestamp value

```
PUBLIC void ixOsServUnload ( void )
```

Used to un-map memory.

**Parameters:**

*None*

**Returns:**

None

```
PUBLIC void ixOsServYield ( void )
```

Yields execution of the current thread.

**Parameters:**

*none*

**Returns:**

none

# IXP425 Performance Profiling (IxPerfProfAcc) API

IXP425 Performance Profiling Utility component Public API.

## Data Structures

- struct **IxPerfProfAccBusPmuResults**  
*Results obtained from running the Bus Pmu component. The results are obtained when the get functions is called.*
- struct **IxPerfProfAccXcycleResults**  
*Results obtained from Xcycle run.*
- struct **IxPerfProfAccXscalePmuEvtCnt**  
*contains results of a counter*
- struct **IxPerfProfAccXscalePmuResults**  
*contains results of counters and their overflow*
- struct **IxPerfProfAccXscalePmuSamplePcProfile**  
*contains summary of samples taken*

## Defines

- #define **IX\_PERFPROF\_ACC\_XSCALE\_PMU\_MAX\_PROFILE\_SAMPLES**  
*This is the maximum number of profiling samples allowed, which can be modified according to user's discretion.*
- #define **IX\_PERFPROF\_ACC\_BUS\_PMU\_MAX\_PECs**  
*This is the maximum number of Programmable Event Counters available. This is a hardware specific and fixed value. Do not change.*
- #define **IX\_PERFPROF\_ACC\_XCYCLE\_MAX\_NUM\_OF\_MEASUREMENTS**  
*Max number of measurement allowed. This constant is used when creating storage array for Xcycle. When run in continuous mode, Xcycle will wrap around and re-use buffer.*
- #define **IX\_PERFPROF\_ACC\_LOG**(level, str, a, b, c, d, e, f)  
*Mechanism for logging a formatted message for the PerfProfAcc component.*

## Enumerations

- enum **IxPerfProfAccBusPmuEventCounters1** {  
    **IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC1\_NORTH\_NPEA\_GRANT\_SELECT**,  
    **IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC1\_NORTH\_NPEB\_GRANT\_SELECT**,  
    **IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC1\_NORTH\_NPEC\_GRANT\_SELECT**,

```

IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_BUS_IDLE_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_NPEA_REQ_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_NPEB_REQ_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_NPEC_REQ_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_GSKT_GRANT_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_ABB_GRANT_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_PCI_GRANT_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_APB_GRANT_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_GSKT_REQ_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_ABB_REQ_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_PCI_REQ_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_APB_REQ_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_0_HIT_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_1_HIT_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_2_HIT_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_3_HIT_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_4_MISS_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_5_MISS_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_6_MISS_SELECT,
IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_7_MISS_SELECT
}

```

*Type of bus pmu events supported on PEC 1.*

```

enum IxPerfProfAccBusPmuEventCounters2 {
    IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEA_XFER_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEB_XFER_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEC_XFER_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_BUS_WRITE_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEA_OWN_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEB_OWN_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEC_OWN_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_GSKT_XFER_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_ABB_XFER_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_PCI_XFER_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_APB_XFER_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_GSKT_OWN_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_ABB_OWN_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_PCI_OWN_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_APB_OWN_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_1_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_2_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_3_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_4_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_5_MISS_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_6_MISS_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_7_MISS_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_0_MISS_SELECT
}

```

*Type of bus pmu events supported on PEC 2.*

enum

```

IxPerfProfAccBusPmuEventCounters3 {
    IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEA_RETRY_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEB_RETRY_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEC_RETRY_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_BUS_READ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEA_WRITE_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEB_WRITE_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEC_WRITE_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_GSKT_RETRY_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_ABB_RETRY_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_PCI_RETRY_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_APB_RETRY_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_GSKT_WRITE_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_ABB_WRITE_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_PCI_WRITE_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_APB_WRITE_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_2_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_3_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_4_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_5_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_6_MISS_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_7_MISS_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_0_MISS_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_1_MISS_SELECT
}

```

*Type of bus pmu events supported on PEC 3.*

```

enum IxPerfProfAccBusPmuEventCounters4 {
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_PCI_SPLIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_EXP_SPLIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_APB_GRANT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_APB_XFER_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_GSKT_READ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_ABB_READ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_PCI_READ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_APB_READ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_NORTH_ABB_SPLIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_NORTH_NPEA_REQ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_NORTH_NPEA_READ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_NORTH_NPEB_READ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_NORTH_NPEC_READ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_3_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_4_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_5_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_6_HIT_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_7_MISS_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_0_MISS_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_1_MISS_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_2_MISS_SELECT
}

```



*Type of bus pmu events supported on PEC 4.*

```
enum IxPerfProfAccBusPmuEventCounters5 {  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_ABB_GRANT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_ABB_XFER_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_ABB_RETRY_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_EXP_SPLIT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_ABB_REQ_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_ABB_OWN_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_BUS_IDLE_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_GRANT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_XFER_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_RETRY_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_REQ_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_OWN_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_WRITE_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_READ_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_4_HIT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_5_HIT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_6_HIT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_7_HIT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_0_MISS_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_1_MISS_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_2_MISS_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_3_MISS_SELECT  
}
```

*Type of bus pmu events supported on PEC 5.*

```
enum IxPerfProfAccBusPmuEventCounters6 {  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SOUTH_PCI_GRANT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SOUTH_PCI_XFER_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SOUTH_PCI_RETRY_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SOUTH_PCI_SPLIT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SOUTH_PCI_REQ_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SOUTH_PCI_OWN_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SOUTH_BUS_WRITE_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_NORTH_NPEC_GRANT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_NORTH_NPEC_XFER_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_NORTH_NPEC_RETRY_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_NORTH_NPEC_REQ_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_NORTH_NPEC_OWN_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_NORTH_NPEB_WRITE_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_NORTH_NPEC_READ_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_5_HIT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_6_HIT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_7_HIT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_0_HIT_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_1_MISS_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_2_MISS_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_3_MISS_SELECT,  
    IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_4_MISS_SELECT
```

```
}
Type of bus pmu events supported on PEC 6.
```

```
enum IxPerfProfAccBusPmuEventCounters7 {
    IX_PERFPROF_ACC_BUS_PMU_PEC7_SOUTH_APB_RETRY_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC7_SOUTH_APB_REQ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC7_SOUTH_APB_OWN_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC7_SOUTH_BUS_READ_SELECT,
    IX_PERFPROF_ACC_BUS_PMU_PEC7_CYCLE_COUNT_SELECT
}
Type of bus pmu events supported on PEC 7.
```

```
enum IxPerfProfAccXscalePmuEvent {
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_CACHE_MISS,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_CACHE_INSTRUCTION,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_STALL,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_INST_TLB_MISS,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_DATA_TLB_MISS,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_BRANCH_EXEC,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_BRANCH_MISPREDICT,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_INST_EXEC,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_FULL_EVERYCYCLE,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_ONCE,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_DATA_CACHE_ACCESS,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_DATA_CACHE_MISS,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_DATA_CACHE_WRITEBACK,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_SW_CHANGE_PC,
    IX_PERFPROF_ACC_XSCALE_PMU_EVENT_MAX
}
Type of xscale pmu events supported.
```

```
enum IxPerfProfAccStatus {
    IX_PERFPROF_ACC_STATUS_SUCCESS,
    IX_PERFPROF_ACC_STATUS_FAIL,
    IX_PERFPROF_ACC_STATUS_ANOTHER_UTIL_IN_PROGRESS,
    IX_PERFPROF_ACC_STATUS_XCYCLE_MEASUREMENT_IN_PROGRESS,
    IX_PERFPROF_ACC_STATUS_XCYCLE_NO_BASELINE,
    IX_PERFPROF_ACC_STATUS_XCYCLE_MEASUREMENT_REQUEST_OUT_OF_RANGE,
    IX_PERFPROF_ACC_STATUS_XCYCLE_PRIORITY_SET_FAIL,
    IX_PERFPROF_ACC_STATUS_XCYCLE_THREAD_CREATE_FAIL,
    IX_PERFPROF_ACC_STATUS_XCYCLE_PRIORITY_RESTORE_FAIL,
    IX_PERFPROF_ACC_STATUS_XCYCLE_MEASUREMENT_NOT_RUNNING,
    IX_PERFPROF_ACC_STATUS_XSCALE_PMU_NUM_INVALID,
    IX_PERFPROF_ACC_STATUS_XSCALE_PMU_EVENT_INVALID,
    IX_PERFPROF_ACC_STATUS_XSCALE_PMU_START_NOT_CALLED,
    IX_PERFPROF_ACC_STATUS_BUS_PMU_MODE_ERROR,
    IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC1_ERROR,
    IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC2_ERROR,
    IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC3_ERROR,
    IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC4_ERROR,
    IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC5_ERROR,
```

```

    IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC6_ERROR,
    IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC7_ERROR,
    IX_PERFPROF_ACC_STATUS_BUS_PMU_START_NOT_CALLED
}
Invalid Status Definitions.

```

```

enum IxPerfProfAccBusPmuMode {
    IX_PERFPROF_ACC_BUS_PMU_MODE_HALT,
    IX_PERFPROF_ACC_BUS_PMU_MODE_SOUTH,
    IX_PERFPROF_ACC_BUS_PMU_MODE_NORTH,
    IX_PERFPROF_ACC_BUS_PMU_MODE_SDRAM
}
State selection of counters.

```

## Functions

```

PUBLICIxPerfProfAccXscalePmuEventCountStart (BOOL clkCntDiv, UINT32 numEvents,
IxPerfProfAccStatus IxPerfProfAccXscalePmuEvent pmuEvent1, IxPerfProfAccXscalePmuEvent pmuEvent2,
IxPerfProfAccXscalePmuEvent pmuEvent3, IxPerfProfAccXscalePmuEvent pmuEvent4)
This API will start the clock and event counting.

```

```

PUBLICIxPerfProfAccXscalePmuEventCountStop (IxPerfProfAccXscalePmuResults
IxPerfProfAccStatus *eventCountStopResults)
This API will stop the clock and event counting.

```

```

PUBLIC
IxPerfProfAccStatus ixPerfProfAccXscalePmuTimeSampStart (UINT32 samplingRate, BOOL clkCntDiv)
Starts the time based sampling.

```

```

PUBLICIxPerfProfAccXscalePmuTimeSampStop (IxPerfProfAccXscalePmuEvtCnt *clkCount,
IxPerfProfAccStatus IxPerfProfAccXscalePmuSamplePcProfile *timeProfile)
Stops the time based sampling.

```

```

PUBLICIxPerfProfAccXscalePmuEventSampStart (UINT32 numEvents, IxPerfProfAccXscalePmu
IxPerfProfAccStatus pmuEvent1, UINT32 eventRate1, IxPerfProfAccXscalePmuEvent pmuEvent2, UINT32 event
IxPerfProfAccXscalePmuEvent pmuEvent3, UINT32 eventRate3, IxPerfProfAccXscalePmu
pmuEvent4, UINT32 eventRate4)
Starts the event based sampling.

```

```

PUBLICIxPerfProfAccXscalePmuEventSampStop (IxPerfProfAccXscalePmuSamplePcProfile
IxPerfProfAccStatus *eventProfile1, IxPerfProfAccXscalePmuSamplePcProfile *eventProfile2,
IxPerfProfAccXscalePmuSamplePcProfile *eventProfile3,
IxPerfProfAccXscalePmuSamplePcProfile *eventProfile4)
Stops the event based sampling.

```

```

PUBLIC void ixPerfProfAccXscalePmuResultsGet (IxPerfProfAccXscalePmuResults *results)
Reads the current value of the counters and their overflow.

```

**PUBLIC** **ixPerfProfAccBusPmuStart** (**ixPerfProfAccBusPmuMode** mode,  
**ixPerfProfAccStatus ixPerfProfAccBusPmuEventCounters1** pecEvent1, **ixPerfProfAccBusPmuEventCounters2**  
 pecEvent2, **ixPerfProfAccBusPmuEventCounters3** pecEvent3,  
**ixPerfProfAccBusPmuEventCounters4** pecEvent4, **ixPerfProfAccBusPmuEventCounters5**  
 pecEvent5, **ixPerfProfAccBusPmuEventCounters6** pecEvent6,  
**ixPerfProfAccBusPmuEventCounters7** pecEvent7)  
*Initializes all the counters and selects events to be monitored.*

**PUBLIC**  
**ixPerfProfAccStatus ixPerfProfAccBusPmuStop** (void)  
*Stops all counters.*

**PUBLIC** void **ixPerfProfAccBusPmuResultsGet** (**ixPerfProfAccBusPmuResults** \*BusPmuResults)  
*Gets values of all counters.*

**PUBLIC** void **ixPerfProfAccBusPmuPMSRGet** (UINT32 \*pmsrValue)  
*Get values of PMSR.*

**PUBLIC**  
**ixPerfProfAccStatus ixPerfProfAccXcycleBaselineRun** (UINT32 \*numBaselineCycle)  
*Perform baseline for Xcycle.*

**PUBLIC**  
**ixPerfProfAccStatus ixPerfProfAccXcycleStart** (UINT32 numMeasurementsRequested)  
*Start the measurement.*

**PUBLIC**  
**ixPerfProfAccStatus ixPerfProfAccXcycleStop** (void)  
*Stop the Xcycle measurement.*

**PUBLIC**  
**ixPerfProfAccStatus ixPerfProfAccXcycleResultsGet** (**ixPerfProfAccXcycleResults** \*xcycleResult)  
*Get the results of Xcycle measurement.*

**PUBLIC** **BOOL ixPerfProfAccXcycleInProgress** (void)  
*Check if Xcycle is running.*

int **ixPerfProfAccXscalePmuTimeSampCreateProcFile** (char \*buf, char \*\*start, off\_t offset, int  
 int \*eof, void \*data)  
 int **ixPerfProfAccXscalePmuEventSampCreateProcFile** (char \*buf, char \*\*start, off\_t offset, int  
 int \*eof, void \*data)

---

## Detailed Description

IXP425 Performance Profiling Utility component Public API.

---

## Define Documentation

```
#define IX_PERFPROF_ACC_BUS_PMU_MAX_PPCS
```

This is the maximum number of Programmable Event Counters available. This is a hardware specific and fixed value. Do not change.

Definition at line **85** of file **IxPerfProfAcc.h**.

```
#define IX_PERFPROF_ACC_LOG ( level,  
                                str,  
                                a,  
                                b,  
                                c,  
                                d,  
                                e,  
                                f      )
```

Mechanism for logging a formatted message for the PerfProfAcc component.

### ***Parameters:***

*UINT32* [in] level – trace level  
*char\** [in] str – format string, similar to printf().  
*UINT32* [in] a – first argument to display  
*UINT32* [in] b – second argument to display  
*UINT32* [in] c – third argument to display  
*UINT32* [in] d – fourth argument to display  
*UINT32* [in] e – fifth argument to display  
*UINT32* [in] f – sixth argument to display

### ***Returns:***

none

Definition at line **117** of file **IxPerfProfAcc.h**.

```
#define IX_PERFPROF_ACC_XCYCLE_MAX_NUM_OF_MEASUREMENTS
```

Max number of measurement allowed. This constant is used when creating storage array for Xcycle. When run in continuous mode, Xcycle will wrap around and re-use buffer.

Definition at line **96** of file **IxPerfProfAcc.h**.

```
#define IX_PERFPROF_ACC_XSCALE_PMU_MAX_PROFILE_SAMPLES
```

This is the maximum number of profiling samples allowed, which can be modified according to the user's discretion.

Definition at line 74 of file **IxPerfProfAcc.h**.

---

## Enumeration Type Documentation

```
enum IxPerfProfAccBusPmuEventCounters1
```

Type of bus pmu events supported on PEC 1.

Lists all bus pmu events.

### *Enumeration values:*

<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_NPEA_GRANT_SELECT</i>	Select North NPEA grant on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_NPEB_GRANT_SELECT</i>	Select North NPEB grant on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_NPEC_GRANT_SELECT</i>	Select North NPEC grant on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_BUS_IDLE_SELECT</i>	Select North bus idle on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_NPEA_REQ_SELECT</i>	Select North NPEA req on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_NPEB_REQ_SELECT</i>	Select North NPEB req on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_NORTH_NPEC_REQ_SELECT</i>	Select North NPEC req on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_GSKT_GRANT_SELECT</i>	Select south gasket grant on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_ABB_GRANT_SELECT</i>	Select south abb grant on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_PCI_GRANT_SELECT</i>	Select south pci grant on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_APB_GRANT_SELECT</i>	Select south apb grant on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_GSKT_REQ_SELECT</i>	Select south gasket request on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_ABB_REQ_SELECT</i>	Select south abb request on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_PCI_REQ_SELECT</i>	Select south pci request on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SOUTH_APB_REQ_SELECT</i>	Select south apb request on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_0_HIT_SELECT</i>	Select sdram0 hit on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_1_HIT_SELECT</i>	

<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_2_HIT_SELECT</i>	Select sdram1 hit on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_3_HIT_SELECT</i>	Select sdram2 hit on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_4_MISS_SELECT</i>	Select sdram3 hit on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_5_MISS_SELECT</i>	Select sdram4 miss on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_6_MISS_SELECT</i>	Select sdram5 miss on PEC1.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC1_SDR_7_MISS_SELECT</i>	Select sdram6 miss on PEC1.
	Select sdram7 miss on PEC1.

Definition at line **208** of file **IxPerfProfAcc.h**.

```
enum IxPerfProfAccBusPmuEventCounters2
```

Type of bus pmu events supported on PEC 2.

Lists all bus pmu events.

**Enumeration values:**

<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEA_XFER_SELECT</i>	Select North NPEA transfer on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEB_XFER_SELECT</i>	Select North NPEB transfer on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEC_XFER_SELECT</i>	Select North NPEC transfer on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_BUS_WRITE_SELECT</i>	Select North bus write on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEA_OWN_SELECT</i>	Select North NPEA own on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEB_OWN_SELECT</i>	Select North NPEB own on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_NORTH_NPEC_OWN_SELECT</i>	Select North NPEC own on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_GSKT_XFER_SELECT</i>	Select South gasket transfer on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_ABB_XFER_SELECT</i>	Select South abb transfer on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_PCI_XFER_SELECT</i>	Select South pci transfer on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_APB_XFER_SELECT</i>	Select South apb transfer on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_GSKT_OWN_SELECT</i>	Select South gasket own on PEC2.

<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_ABB_OWN_SELECT</i>	Select South abb own on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_PCI_OWN_SELECT</i>	Select South pci own on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SOUTH_APB_OWN_SELECT</i>	Select South apb own transfer on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_1_HIT_SELECT</i>	Select sdram1 hit on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_2_HIT_SELECT</i>	Select sdram2 hit on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_3_HIT_SELECT</i>	Select sdram3 hit on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_4_HIT_SELECT</i>	Select sdram4 hit on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_5_MISS_SELECT</i>	Select sdram5 miss on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_6_MISS_SELECT</i>	Select sdram6 miss on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_7_MISS_SELECT</i>	Select sdram7 miss on PEC2.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC2_SDR_0_MISS_SELECT</i>	Select sdram0 miss on PEC2.

Definition at line **246** of file **IxPerfProfAcc.h**.

```
enum IxPerfProfAccBusPmuEventCounters3
```

Type of bus pmu events supported on PEC 3.

Lists all bus pmu events.

**Enumeration values:**

<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEA_RETRY_SELECT</i>	Select north NPEA retry on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEB_RETRY_SELECT</i>	Select north NPEB retry on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEC_RETRY_SELECT</i>	Select north NPEC retry on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_BUS_READ_SELECT</i>	Select north bus read on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEA_WRITE_SELECT</i>	Select north NPEA write on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEB_WRITE_SELECT</i>	Select north NPEB write on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_NORTH_NPEC_WRITE_SELECT</i>	Select north NPEC write on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_GSKT_RETRY_SELECT</i>	Select south gasket retry on PEC3.



<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_ABB_RETRY_SELECT</i>	Select south abb retry on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_PCI_RETRY_SELECT</i>	Select south pci retry on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_APB_RETRY_SELECT</i>	Select south apb retry on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_GSKT_WRITE_SELECT</i>	Select south gasket write on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_ABB_WRITE_SELECT</i>	Select south abb write on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_PCI_WRITE_SELECT</i>	Select south pci write on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SOUTH_APB_WRITE_SELECT</i>	Select south apb write on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_2_HIT_SELECT</i>	Select sdram2 hit on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_3_HIT_SELECT</i>	Select sdram3 hit on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_4_HIT_SELECT</i>	Select sdram4 hit on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_5_HIT_SELECT</i>	Select sdram5 hit on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_6_MISS_SELECT</i>	Select sdram6 miss on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_7_MISS_SELECT</i>	Select sdram7 miss on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_0_MISS_SELECT</i>	Select sdram0 miss on PEC3.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC3_SDR_1_MISS_SELECT</i>	Select sdram1 miss on PEC3.

Definition at line **284** of file **IxPerfProfAcc.h**.

```
enum IxPerfProfAccBusPmuEventCounters4
```

Type of bus pmu events supported on PEC 4.

Lists all bus pmu events.

**Enumeration values:**

<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_PCI_SPLIT_SELECT</i>	Select south pci split on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_EXP_SPLIT_SELECT</i>	Select south expansion split on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_APB_GRANT_SELECT</i>	Select south apb grant on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_APB_XFER_SELECT</i>	

	Select south apb transfer on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_GSKT_READ_SELECT</i>	Select south gasket read on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_ABB_READ_SELECT</i>	Select south abb read on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_PCI_READ_SELECT</i>	Select south pci read on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SOUTH_APB_READ_SELECT</i>	Select south apb read on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_NORTH_ABB_SPLIT_SELECT</i>	Select north abb split on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_NORTH_NPEA_REQ_SELECT</i>	Select north NPEA req on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_NORTH_NPEA_READ_SELECT</i>	Select north NPEA read on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_NORTH_NPEB_READ_SELECT</i>	Select north NPEB read on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_NORTH_NPEC_READ_SELECT</i>	Select north NPEC read on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_3_HIT_SELECT</i>	Select sdram3 hit on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_4_HIT_SELECT</i>	Select sdram4 hit on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_5_HIT_SELECT</i>	Select sdram5 hit on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_6_HIT_SELECT</i>	Select sdram6 hit on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_7_MISS_SELECT</i>	Select sdram7 miss on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_0_MISS_SELECT</i>	Select sdram0 miss on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_1_MISS_SELECT</i>	Select sdram1 miss on PEC4.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC4_SDR_2_MISS_SELECT</i>	Select sdram2 miss on PEC4.

Definition at line **322** of file **IxPerfProfAcc.h**.

```
enum IxPerfProfAccBusPmuEventCounters5
```

Type of bus pmu events supported on PEC 5.

Lists all bus pmu events.

**Enumeration values:**

<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_ABB_GRANT_SELECT</i>	Select south abb grant on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_ABB_XFER_SELECT</i>	Select south abb transfer on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_ABB_RETRY_SELECT</i>	Select south abb retry on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_EXP_SPLIT_SELECT</i>	Select south expansion split on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_ABB_REQ_SELECT</i>	Select south abb request on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_ABB_OWN_SELECT</i>	Select south abb own on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SOUTH_BUS_IDLE_SELECT</i>	Select south bus idle on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_GRANT_SELECT</i>	Select north NPEB grant on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_XFER_SELECT</i>	Select north NPEB transfer on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_RETRY_SELECT</i>	Select north NPEB retry on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_REQ_SELECT</i>	Select north NPEB request on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_OWN_SELECT</i>	Select north NPEB own on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_WRITE_SELECT</i>	Select north NPEB write on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_NORTH_NPEB_READ_SELECT</i>	Select north NPEB read on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_4_HIT_SELECT</i>	Select north sdram4 hit on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_5_HIT_SELECT</i>	Select north sdram5 hit on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_6_HIT_SELECT</i>	Select north sdram6 hit on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_7_HIT_SELECT</i>	Select north sdram7 hit on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_0_MISS_SELECT</i>	Select north sdram0 miss on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_1_MISS_SELECT</i>	Select north sdram1 miss on PEC5.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC5_SDR_2_MISS_SELECT</i>	Select north sdram2 miss on PEC5.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC5\_SDR\_3\_MISS\_SELECT*

Select north sdram3 miss on PEC5.

Definition at line **358** of file **IxPerfProfAcc.h**.

enum IxPerfProfAccBusPmuEventCounters6

Type of bus pmu events supported on PEC 6.

Lists all bus pmu events.

**Enumeration values:**

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SOUTH\_PCI\_GRANT\_SELECT*

Select south pci grant on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SOUTH\_PCI\_XFER\_SELECT*

Select south pci transfer on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SOUTH\_PCI\_RETRY\_SELECT*

Select south pci retry on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SOUTH\_PCI\_SPLIT\_SELECT*

Select south pci split on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SOUTH\_PCI\_REQ\_SELECT*

Select south pci request on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SOUTH\_PCI\_OWN\_SELECT*

Select south pci own on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SOUTH\_BUS\_WRITE\_SELECT*

Select south pci write on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_NORTH\_NPEC\_GRANT\_SELECT*

Select north NPEC grant on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_NORTH\_NPEC\_XFER\_SELECT*

Select north NPEC transfer on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_NORTH\_NPEC\_RETRY\_SELECT*

Select north NPEC retry on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_NORTH\_NPEC\_REQ\_SELECT*

Select north NPEC request on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_NORTH\_NPEC\_OWN\_SELECT*

Select north NPEC own on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_NORTH\_NPEB\_WRITE\_SELECT*

Select north NPEB write on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_NORTH\_NPEC\_READ\_SELECT*

Select north NPEC read on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SDR\_5\_HIT\_SELECT*

Select sdram5 hit on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SDR\_6\_HIT\_SELECT*

Select sdram6 hit on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SDR\_7\_HIT\_SELECT*

Select sdram7 hit on PEC6.

*IX\_PERFPROF\_ACC\_BUS\_PMU\_PEC6\_SDR\_0\_HIT\_SELECT*

Select sdram0 hit on PEC6.

<i>IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_1_MISS_SELECT</i>	Select sdram1 miss on PEC6.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_2_MISS_SELECT</i>	Select sdram2 miss on PEC6.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_3_MISS_SELECT</i>	Select sdram3 miss on PEC6.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC6_SDR_4_MISS_SELECT</i>	Select sdram4 miss on PEC6.

Definition at line **395** of file **IxPerfProfAcc.h**.

```
enum IxPerfProfAccBusPmuEventCounters7
```

Type of bus pmu events supported on PEC 7.

Lists all bus pmu events.

**Enumeration values:**

<i>IX_PERFPROF_ACC_BUS_PMU_PEC7_SOUTH_APB_RETRY_SELECT</i>	Select south apb retry on PEC7.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC7_SOUTH_APB_REQ_SELECT</i>	Select south apb request on PEC7.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC7_SOUTH_APB_OWN_SELECT</i>	Select south apb own on PEC7.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC7_SOUTH_BUS_READ_SELECT</i>	Select south bus read on PEC7.
<i>IX_PERFPROF_ACC_BUS_PMU_PEC7_CYCLE_COUNT_SELECT</i>	Select cycle count on PEC7.

Definition at line **432** of file **IxPerfProfAcc.h**.

```
enum IxPerfProfAccBusPmuMode
```

State selection of counters.

These states will be used to determine the counters whose values are to be read.

**Enumeration values:**

<i>IX_PERFPROF_ACC_BUS_PMU_MODE_HALT</i>	halt state
<i>IX_PERFPROF_ACC_BUS_PMU_MODE_SOUTH</i>	south state
<i>IX_PERFPROF_ACC_BUS_PMU_MODE_NORTH</i>	north state
<i>IX_PERFPROF_ACC_BUS_PMU_MODE_SDRAM</i>	SDRAM state.

Definition at line **553** of file **IxPerfProfAcc.h**.

```
enum IxPerfProfAccStatus
```

## Invalid Status Definitions.

These status will be used by the APIs to return to the user.

### **Enumeration values:**

<i>IX_PERFPROF_ACC_STATUS_SUCCESS</i>	success
<i>IX_PERFPROF_ACC_STATUS_FAIL</i>	fail
<i>IX_PERFPROF_ACC_STATUS_ANOTHER_UTIL_IN_PROGRESS</i>	another utility in progress
<i>IX_PERFPROF_ACC_STATUS_XCYCLE_MEASUREMENT_IN_PROGRESS</i>	measurement in progress
<i>IX_PERFPROF_ACC_STATUS_XCYCLE_NO_BASELINE</i>	no baseline yet
<i>IX_PERFPROF_ACC_STATUS_XCYCLE_MEASUREMENT_REQUEST_OUT_OF_RANGE</i>	Measurement chosen is out of range.
<i>IX_PERFPROF_ACC_STATUS_XCYCLE_PRIORITY_SET_FAIL</i>	Cannot set task priority.
<i>IX_PERFPROF_ACC_STATUS_XCYCLE_THREAD_CREATE_FAIL</i>	Fail create thread.
<i>IX_PERFPROF_ACC_STATUS_XCYCLE_PRIORITY_RESTORE_FAIL</i>	cannot restore priority
<i>IX_PERFPROF_ACC_STATUS_XCYCLE_MEASUREMENT_NOT_RUNNING</i>	xcycle not running
<i>IX_PERFPROF_ACC_STATUS_XSCALE_PMU_NUM_INVALID</i>	invalid number entered
<i>IX_PERFPROF_ACC_STATUS_XSCALE_PMU_EVENT_INVALID</i>	invalid pmu event
<i>IX_PERFPROF_ACC_STATUS_XSCALE_PMU_START_NOT_CALLED</i>	a start process was not called before attempting a stop or results get
<i>IX_PERFPROF_ACC_STATUS_BUS_PMU_MODE_ERROR</i>	invalid mode
<i>IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC1_ERROR</i>	invalid pec1 entered
<i>IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC2_ERROR</i>	invalid pec2 entered
<i>IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC3_ERROR</i>	invalid pec3 entered
<i>IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC4_ERROR</i>	invalid pec4 entered
<i>IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC5_ERROR</i>	invalid pec5 entered
<i>IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC6_ERROR</i>	invalid pec6 entered
<i>IX_PERFPROF_ACC_STATUS_BUS_PMU_PEC7_ERROR</i>	

invalid pec7  
entered  
a start process  
was not called  
before  
attempting a  
stop

*IX\_PERFPROF\_ACC\_STATUS\_BUS\_PMU\_START\_NOT\_CALLED*

Definition at line **491** of file **IxPerfProfAcc.h**.

enum IxPerfProfAccXscalePmuEvent

Type of xscale pmu events supported.

Lists all xscale pmu events. The maximum is a default value that the user should not exceed.

**Enumeration values:**

<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_CACHE_MISS</i>	cache miss
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_CACHE_INSTRUCTION</i>	cache instruction
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_STALL</i>	event stall
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_INST_TLB_MISS</i>	instruction tlb miss
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_DATA_TLB_MISS</i>	data tlb miss
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_BRANCH_EXEC</i>	branch executed
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_BRANCH_MISPREDICT</i>	branch mispredict
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_INST_EXEC</i>	instruction executed
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_FULL_EVERYCYCLE</i>	Stall – data cache buffers are full.  This event occurs every cycle where condition present
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_ONCE</i>	Stall – data cache buffers are full.This event occurs once for each contiguous sequence.
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_DATA_CACHE_ACCESS</i>	data cache access
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_DATA_CACHE_MISS</i>	data cache miss
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_DATA_CACHE_WRITEBACK</i>	data cache writeback
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_SW_CHANGE_PC</i>	sw change pc
<i>IX_PERFPROF_ACC_XSCALE_PMU_EVENT_MAX</i>	max value

Definition at line **451** of file **IxPerfProfAcc.h**.

# Function Documentation

ixPerfProfAccBusPmuPMSRGet ( UINT32 \* *pmsrValue* )

Get values of PMSR.

This API gets the Previous Master Slave Register value and returns it to the calling function. This value indicates which master or slave accessed the north, south bus or sdram last. The value returned by this function is a 32 bit value and is read from location of an offset 0x0024 of the base value.

The PMSR value returned indicate the following:

```
*****
*   Bit       *   Name   *   Description                               *
*   *         *   *         *   *
*****
* [31:18] *Reserved*
*****
* [17:12] * PSS      * Indicates which of the slaves on
*         *         * ARBS was previously
*         *         * accessed by the AHBS.
*         *         * [000001] Expansion Bus
*         *         * [000010] SDRAM Controller
*         *         * [000100] PCI
*         *         * [001000] Queue Manager
*         *         * [010000] AHB-APB Bridge
*         *         * [100000] Reserved
*****
* [11:8]   * PSN      * Indicates which of the Slaves on
*         *         * ARBN was previously
*         *         * accessed the AHBN.
*         *         * [0001] SDRAM Controller
*         *         * [0010] AHB-AHB Bridge
*         *         * [0100] Reserved
*         *         * [1000] Reserved
*****
* [7:4]    * PMS      * Indicates which of the Masters on
*         *         * ARBS was previously
*         *         * accessing the AHBS.
*         *         * [0001] Gasket
*         *         * [0010] AHB-AHB Bridge
*         *         * [0100] PCI
*         *         * [1000] APB
*****
* [3:0]    * PMN      * Indicates which of the Masters on
*         *         * ARBN was previously
*         *         * accessing the AHBN.
*         *         * [0001] NPEA
*         *         * [0010] NPEB
*         *         * [0100] NPEC
*         *         * [1000] Reserved
*****
```

**Parameters:**



*UINT32 \*psmrValue* – Pointer to return PMSR value. Users need to allocate storage for psmrValue.

**Returns:**

none

- Reentrant : no
- ISR Callable : no

```
ixPerfProfAccBusPmuResultsGet ( IxPerfProfAccBusPmuResults * BusPmuResults )
```

Gets values of all counters.

This function is responsible for getting all the counter values from the lower API and putting it into an array for the user.

**Parameters:**

- "IxPerfProfAccBusPmuResults \*busPmuResults"  
    ◇ Pointer to a structure of arrays to store all counter values.

**Returns:**

none

- Reentrant : no
- ISR Callable : no

```
ixPerfProfAccBusPmuStart ( IxPerfProfAccBusPmuMode                    mode,  
                          IxPerfProfAccBusPmuEventCounters1 pecEvent1,  
                          IxPerfProfAccBusPmuEventCounters2 pecEvent2,  
                          IxPerfProfAccBusPmuEventCounters3 pecEvent3,  
                          IxPerfProfAccBusPmuEventCounters4 pecEvent4,  
                          IxPerfProfAccBusPmuEventCounters5 pecEvent5,  
                          IxPerfProfAccBusPmuEventCounters6 pecEvent6,  
                          IxPerfProfAccBusPmuEventCounters7 pecEvent7  
                          )
```

Initializes all the counters and selects events to be monitored.

Function initializes all the counters and assigns the events associated with the counters. Users send in the mode and events they want to count. This API verifies if the combination chosen is appropriate and sets all the registers accordingly. Selecting HALT mode will result in an error. User should use **ixPerfProfAccBusPmuStop()** to HALT.

**Parameters:**

- IxPerfProfAccStateBusPmuMode*            mode – Mode selection.
- IxPerfProfAccBusPmuEventCounters1*    pecEvent1 – Event for PEC1.
- IxPerfProfAccBusPmuEventCounters2*    pecEvent2 – Event for PEC2.

*IxPerfProfAccBusPmuEventCounters3* pecEvent3 – Event for PEC3.  
*IxPerfProfAccBusPmuEventCounters4* pecEvent4 – Event for PEC4.  
*IxPerfProfAccBusPmuEventCounters5* pecEvent5 – Event for PEC5.  
*IxPerfProfAccBusPmuEventCounters6* pecEvent6 – Event for PEC6.  
*IxPerfProfAccBusPmuEventCounters7* pecEvent7 – Event for PEC7.

**Returns:**

- ◇ IX\_PERFPROF\_ACC\_STATUS\_SUCCESS – Initialization executed successfully.
- ◇ IX\_PERFPROF\_ACC\_STATUS\_BUS\_PMU\_MODE\_ERROR – Error in selection of mode. Only NORTH, SOUTH and SDRAM modes are allowed.
- ◇ IX\_PERFPROF\_ACC\_STATUS\_BUS\_PMU\_PEC1\_ERROR – Error in selection of event for PEC1
- ◇ IX\_PERFPROF\_ACC\_STATUS\_BUS\_PMU\_PEC2\_ERROR – Error in selection of event for PEC2
- ◇ IX\_PERFPROF\_ACC\_STATUS\_BUS\_PMU\_PEC3\_ERROR – Error in selection of event for PEC3
- ◇ IX\_PERFPROF\_ACC\_STATUS\_BUS\_PMU\_PEC4\_ERROR – Error in selection of event for PEC4
- ◇ IX\_PERFPROF\_ACC\_STATUS\_BUS\_PMU\_PEC5\_ERROR – Error in selection of event for PEC5
- ◇ IX\_PERFPROF\_ACC\_STATUS\_BUS\_PMU\_PEC6\_ERROR – Error in selection of event for PEC6
- ◇ IX\_PERFPROF\_ACC\_STATUS\_BUS\_PMU\_PEC7\_ERROR – Error in selection of event for PEC7
- ◇ IX\_PERFPROF\_ACC\_STATUS\_ANOTHER\_UTIL\_IN\_PROGRESS – another utility is running
- ◇ IX\_PERFPROF\_ACC\_STATUS\_FAIL – Failed to start because interrupt service routine fails to bind.

- Reentrant : no
- ISR Callable : no

**ixPerfProfAccBusPmuStop ( void )**

Stops all counters.

This function stops all the PECs by setting the halt bit in the ESR.

**Returns:**

- ◇ IX\_PERFPROF\_ACC\_STATUS\_SUCCESS – Counters successfully halted.
- ◇ IX\_PERFPROF\_ACC\_STATUS\_FAIL – Counters could't be halted.
- ◇ IX\_PERFPROF\_ACC\_STATUS\_BUS\_PMU\_START\_NOT\_CALLED – the **ixPerfProfAccBusPmuStart()** function is not called.

- Reentrant : no
- ISR Callable : no

**ixPerfProfAccXcycleBaselineRun ( UINT32 \* numBaselineCycle )**

Perform baseline for Xcycle.

**Parameters:**

*UINT32 [out]* – pointer to baseline value after calibration. Calling function are responsible for allocating memory space for this pointer.  
*\*numBaselineCycle*

Global Data :

- None.

This function MUST be run before the Xcycle tool can be used. This function must be run immediately when the OS boots up with no other addition programs running. Addition note : This API will measure the time needed to perform a fix amount of CPU instructions (~ 1 second worth of loops) as a highest priority task and with interrupt disabled. The time measured is known as the baseline – interpreted as the shortest time needed to complete the amount of CPU instructions. The baseline is returned as unit of time in 66Mhz clock tick.

**Returns:**

- ◇ IX\_PERFPROF\_ACC\_STATUS\_SUCCESS – successful run, result is returned
- ◇ IX\_PERFPROF\_ACC\_STATUS\_XCYCLE\_PRIORITY\_SET\_FAIL – failed to change task priority
- ◇ IX\_PERFPROF\_ACC\_STATUS\_XCYCLE\_PRIORITY\_RESTORE\_FAIL – failed to restore task priority
- ◇ IX\_PERFPROF\_ACC\_STATUS\_ANOTHER\_UTIL\_IN\_PROGRESS – another utility is running
- ◇ IX\_PERFPROF\_ACC\_STATUS\_XCYCLE\_MEASUREMENT\_IN\_PROGRESS – Xcycle tool has already started

- Reentrant : no
- ISR Callable : no

```
ixPerfProfAccXcycleInProgress ( void )
```

Check if Xcycle is running.

**Parameters:**

*None* Global Data :  
◇ None.

Check if Xcycle measuring task is running.

**Returns:**

- ◇ TRUE – Xcycle is running
- ◇ FALSE – Xcycle is not running

- Reentrant : no
- ISR Callable : no

**ixPerfProfAccXcycleResultsGet ( IxPerfProfAccXcycleResults \* xcycleResult )**

Get the results of Xcycle measurement.

**Parameters:**

*IxPerfProfAccXcycleResults [out]*      Pointer to results of last measurements. Calling function are responsible for allocating memory space for this pointer.  
*\*xcycleResult*

Global Data :

- None.

Retrieve the results of last measurement. User should use **ixPerfProfAccXcycleInProgress()** to check if measurement is completed before getting the results.

**Returns:**

◇ IX\_PERFPROF\_ACC\_STATUS\_SUCCESS – successful  
◇ IX\_PERFPROF\_ACC\_STATUS\_FAIL – result is not complete.  
◇ IX\_PERFPROF\_ACC\_STATUS\_XCYCLE\_NO\_BASELINE – baseline is performed  
◇ IX\_PERFPROF\_ACC\_STATUS\_XCYCLE\_MEASUREMENT\_IN\_PROGRESS – Xcycle tool is still running

- Reentrant : no
- ISR Callable : no

**ixPerfProfAccXcycleStart ( UINT32 numMeasurementsRequested )**

Start the measurement.

**Parameters:**

*UINT32 [in]*      – number of measurements to perform. Value can be 0 to  
*numMeasurementsRequested* IX\_PERFPROF\_ACC\_XCYCLE\_MAX\_NUM\_OF\_MEASUREMENTS.  
0 indicate continuous measurement.

Global Data :

- None.

Start the measurements immediately. numMeasurementsRequested specifies number of measurements to run. If numMeasurementsRequested is set to 0, the measurement will be performed continuously until IxPerfProfAccXcycleStop() is called. It is estimated that 1 measurement takes approximately 1 second during low CPU utilization, therefore 128 measurement takes approximately 128 sec. When CPU utilization is high, the measurement will take longer. This function spawn a task the perform the measurement and returns. The measurement may continue even if this function returns.

**IMPORTANT:** Under heavy CPU utilization, the task spawn by this function may starve and fail to respond to stop command. User may need to kill the task manually in this case.

There are only `IX_PERFPROF_ACC_XCYCLE_MAX_NUM_OF_MEASUREMENTS` storage available so storing is wrapped around if measurements are more than `IX_PERFPROF_ACC_XCYCLE_MAX_NUM_OF_MEASUREMENTS`.

**Returns:**

- ◇ `IX_PERFPROF_ACC_STATUS_SUCCESS` – successful start, a thread is created in the background to perform measurement.
- ◇ `IX_PERFPROF_ACC_STATUS_XCYCLE_PRIORITY_SET_FAIL` – failed to set task priority
- ◇ `IX_PERFPROF_ACC_STATUS_XCYCLE_THREAD_CREATE_FAIL` – failed to create thread to perform measurement.
- ◇ `IX_PERFPROF_ACC_STATUS_XCYCLE_NO_BASELINE` – baseline is not available
- ◇ `IX_PERFPROF_ACC_STATUS_XCYCLE_MEASUREMENT_REQUEST_OUT_OF_RANGE` – value is larger than `IX_PERFPROF_ACC_XCYCLE_MAX_NUM_OF_MEASUREMENTS`
- ◇ `IX_PERFPROF_ACC_STATUS_XCYCLE_MEASUREMENT_IN_PROGRESS` – Xcycle tool has already started
- ◇ `IX_PERFPROF_ACC_STATUS_ANOTHER_UTIL_IN_PROGRESS` – another utility is running

- Reentrant : no
- ISR Callable : no

`ixPerfProfAccXcycleStop ( void )`

Stop the Xcycle measurement.

**Parameters:**

None Global Data :  
◇ None.

Stop Xcycle measurements immediately. If the measurements have stopped or not started, return `IX_PERFPROF_STATUS_XCYCLE_MEASUREMENT_NOT_RUNNING`. Note: This function does not stop measurement cold. The measurement thread may need a few seconds to complete the last measurement. User needs to use `ixPerfProfAccXcycleInProgress()` to determine if measurement is indeed completed.

**Returns:**

- ◇ `IX_PERFPROF_ACC_STATUS_SUCCESS` – successful measurement is stopped
- ◇ `IX_PERFPROF_STATUS_XCYCLE_MEASUREMENT_NOT_RUNNING` – no measurement running

- Reentrant : no
- ISR Callable : no

```

ixPerfProfAccXscalePmuEventCountStart ( BOOL                clkCntDiv,
                                         UINT32             numEvents,
                                         IxPerfProfAccXscalePmuEvent pmuEvent1,
                                         IxPerfProfAccXscalePmuEvent pmuEvent2,
                                         IxPerfProfAccXscalePmuEvent pmuEvent3,
                                         IxPerfProfAccXscalePmuEvent pmuEvent4
                                         )

```

This API will start the clock and event counting.

**Parameters:**

<i>BOOL [in] clkCntDiv</i>	– enables/disables the clock divider. When true, the divider is enabled and the clock count will be incremented by one at each 64th processor clock cycle. When false, the divider is disabled and the clock count will be incremented at every processor clock cycle.
<i>UINT32 [in] numEvents</i>	– the number of PMU events that are to be monitored as specified by the user. For clock counting only, this is set to zero.
<i>IxPerfProfAccXscalePmuEvent [in] pmuEvent1</i>	– the specific PMU event to be monitored by counter 1
<i>IxPerfProfAccXscalePmuEvent [in] pmuEvent2</i>	– the specific PMU event to be monitored by counter 2
<i>IxPerfProfAccXscalePmuEvent [in] pmuEvent3</i>	– the specific PMU event to be monitored by counter 3
<i>IxPerfProfAccXscalePmuEvent [in] pmuEvent4</i>	– the specific PMU event to be monitored by counter 4

This API will start the clock and xscale PMU event counting. Up to 4 events can be monitored simultaneously. This API has to be called before ixPerfProfAccXscalePmuEventCountStop can be called.

**Returns:**

- ◇ IX\_PERFPROF\_ACC\_STATUS\_SUCCESS if clock and events counting are started successfully
- ◇ IX\_PERFPROF\_ACC\_STATUS\_FAIL if unable to start the counting
- ◇ IX\_PERFPROF\_ACC\_STATUS\_XSCALE\_PMU\_NUM\_INVALID if the number of events specified is out of the valid range
- ◇ IX\_PERFPROF\_ACC\_STATUS\_XSCALE\_PMU\_EVENT\_INVALID if the value of the PMU event specified does not exist
- ◇ IX\_PERFPROF\_ACC\_STATUS\_ANOTHER\_UTIL\_IN\_PROGRESS – another utility is running

- Reentrant : no
- ISR Callable : no

```

ixPerfProfAccXscalePmuEventCountStop ( IxPerfProfAccXscalePmuResults * eventCountStopResults )

```

This API will stop the clock and event counting.

**Parameters:**

<i>IxPerfProfAccXscalePmuResults [out]</i> <i>*eventCountStopResults</i>	– pointer to struct containing results of counters and their overflow. It is the users's responsibility to allocate the memory for this pointer.
---	--

This API will stop the clock and xscale PMU events that are being counted. The results of the clock and events count will be stored in the pointer allocated by the user. It can only be called once IxPerfProfAccEventCountStart has been called.

**Returns:**

- ◇ IX\_PERFPROF\_ACC\_STATUS\_SUCCESS if clock and events counting are stopped successfully
- ◇ IX\_PERFPROF\_ACC\_STATUS\_XSCALE\_PMU\_START\_NOT\_CALLED if ixPerfProfAccXscalePmuEventCountStart is not called first.

- Reentrant : no
- ISR Callable : no

```

ixPerfProfAccXscalePmuEventSampStart (  UINT32                numEvents,
                                         IxPerfProfAccXscalePmuEvent pmuEvent1,
                                         UINT32                eventRate1,
                                         IxPerfProfAccXscalePmuEvent pmuEvent2,
                                         UINT32                eventRate2,
                                         IxPerfProfAccXscalePmuEvent pmuEvent3,
                                         UINT32                eventRate3,
                                         IxPerfProfAccXscalePmuEvent pmuEvent4,
                                         UINT32                eventRate4
                                         )

```

Starts the event based sampling.

**Parameters:**

- |   |   |
|---|---|
| <i>UINT32 [in] &lt;numEvents&gt;</i>              | – the number of PMU events that are to be monitored as specified by the user. The value should be between 1–4 events at a time.   |
| <i>IxPerfProfAccXscalePmuEvent [in] pmuEvent1</i> | – the specific PMU event to be monitored by counter 1   |
| <i>UINT32 [in] eventRate1</i>                     | – sampling rate of counter 1. The rate is the number of events before a sample taken. If 0 is specified, the the full counter value (0xFFFFFFFF) is used. The rate must not be greater than the full counter value. |
| <i>IxPerfProfAccXscalePmuEvent [in] pmuEvent2</i> | – the specific PMU event to be monitored by counter 2   |
| <i>UINT32 [in] eventRate2</i>                     | – sampling rate of counter 2. The rate is the number of events before a sample taken. If 0 is specified, the full counter value (0xFFFFFFFF) is used. The rate must not be greater than the full counter value.     |
| <i>IxPerfProfAccXscalePmuEvent [in] pmuEvent3</i> | – the specific PMU event to be monitored by counter 3   |
| <i>UINT32 [in] eventRate3</i>                     |   |

*IxPerfProfAccXscalePmuEvent*  
[in] *pmuEvent4*  
UINT32 [in] *eventRate4*

- sampling rate of counter 3. The rate is the number of events before a sample taken. If 0 is specified, the full counter value (0xFFFFFFFF) is used. The rate must not be greater than the full counter value.
- the specific PMU event to be monitored by counter 4
- sampling rate of counter 4. The rate is the number of events before a sample taken. If 0 is specified, the full counter value (0xFFFFFFFF) is used. The rate must not be greater than the full counter value.

Starts the event based sampling to determine the frequency with which events are being executed. The sampling rate is the number of events, as specified by the user, before a counter overflow interrupt is generated. A sample is taken at each counter overflow interrupt. At each sample, the value of the program counter determines the corresponding location in the code. Each of these occurrences are recorded to determine the frequency with which the Xscale code in each event is executed. This API has to be called before `ixPerfProfAccXscalePmuEventSampStop` can be called.

**Returns:**

- ◊ `IX_PERFPROF_ACC_STATUS_SUCCESS` if event based sampling is started successfully
- ◊ `IX_PERFPROF_ACC_STATUS_FAIL` if unable to start the sampling
- ◊ `IX_PERFPROF_ACC_STATUS_XSCALE_PMU_NUM_INVALID` if the number of events specified is out of the valid range
- ◊ `IX_PERFPROF_ACC_STATUS_XSCALE_PMU_EVENT_INVALID` if the value of the PMU event specified does not exist
- `IX_PERFPROF_ACC_STATUS_ANOTHER_UTIL_IN_PROGRESS` – another utility is running

- Reentrant : no
- ISR Callable : no

```
ixPerfProfAccXscalePmuEventSampStop ( IxPerfProfAccXscalePmuSamplePcProfile * eventProfile1,
                                       IxPerfProfAccXscalePmuSamplePcProfile * eventProfile2,
                                       IxPerfProfAccXscalePmuSamplePcProfile * eventProfile3,
                                       IxPerfProfAccXscalePmuSamplePcProfile * eventProfile4
                                       )
```

Stops the event based sampling.

**Parameters:**

- IxPerfProfAccXscalePmuSamplePcProfile* – pointer to the array of profiles for each program counter value;  
[out] *\*eventProfile1* user should set the size of the array to `IX_PERFPROF_ACC_XSCALE_PMU_MAX_PROFILE_SAMPLING_RATE`. It is the users's responsibility to allocate memory for this pointer.
- IxPerfProfAccXscalePmuSamplePcProfile* – pointer to the array of profiles for each program counter value;  
[out] *\*eventProfile2* user should set the size of the array to `IX_PERFPROF_ACC_XSCALE_PMU_MAX_PROFILE_SAMPLING_RATE`. It is the users's responsibility to allocate memory for this pointer.



*IxPerfProfAccXscalePmuSamplePcProfile* – pointer to the array of profiles for each program counter value;  
 [out] \*eventProfile3 user should set the size of the array to  
 IX\_PERFPROF\_ACC\_XSCALE\_PMU\_MAX\_PROFILE\_SAMPLING  
 It is the users's responsibility to allocate memory for this pointer.

*IxPerfProfAccXscalePmuSamplePcProfile* – pointer to the array of profiles for each program counter value;  
 [out] \*eventProfile4 user should set the size of the array to  
 IX\_PERFPROF\_ACC\_XSCALE\_PMU\_MAX\_PROFILE\_SAMPLING  
 It is the users's responsibility to allocate memory for this pointer.

This API stops the event based sampling. The results are stored in the pointers allocated by the user. It can only be called once ixPerfProfAccEventSampStart has been called.

**Returns:**

- ◇ IX\_PERFPROF\_ACC\_STATUS\_SUCCESS if event based sampling is stopped successfully
- ◇ IX\_PERFPROF\_ACC\_STATUS\_XSCALE\_PMU\_START\_NOT\_CALLED if ixPerfProfAccEventSampStart not called first.

- Reentrant : no
- ISR Callable : no

```
ixPerfProfAccXscalePmuResultsGet ( IxPerfProfAccXscalePmuResults * results )
```

Reads the current value of the counters and their overflow.

**Parameters:**

*IxPerfProfAccXscalePmuResults* [out] – pointer to the results struct. It is the user's responsibility  
 \*results to allocate memory for this pointer

This API reads the value of all four event counters and the clock counter, and the associated overflows. It does not give results associated with sampling, i.e. PC and their frequencies. This API can be called at any time once a process has been started. If it is called before a process has started the user should be aware that the values it contains are default values and might be meaningless. The values of the counters are stored in the pointer allocated by the client.

**Returns:**

– none

- Reentrant : no
- ISR Callable : no

```
ixPerfProfAccXscalePmuTimeSampStart ( UINT32 samplingRate,  
                                       BOOL clkCntDiv  
                                       )
```

Starts the time based sampling.

**Parameters:**

<i>UINT32 [in]</i> <i>samplingRate</i>	– sampling rate is the number of clock counts before a counter overflow interrupt is generated, at which, a sample is taken; the rate specified cannot be greater than the counter size of 32bits or set to zero.
<i>BOOL [in]</i> <i>clkCntDiv</i>	– enables/disables the clock divider. When true, the divider is enabled and the clock count will be incremented by one at each 64th processor clock cycle. When false, the divider is disabled and the clock count will be incremented at every processor clock cycle.

This API starts the time based sampling to determine the frequency with which lines of code are being executed. Sampling is done at the rate specified by the user. At each sample, the value of the program counter is determined. Each of these occurrences are recorded to determine the frequency with which the Xscale code is being executed. This API has to be called before ixPerfProfAccXscalePmuTimeSampStop can be called.

**Returns:**

- ◇ IX\_PERFPROF\_ACC\_STATUS\_SUCCESS if time based sampling is started successfully
- ◇ IX\_PERFPROF\_ACC\_STATUS\_FAIL if unable to start the sampling
- ◇ IX\_PERFPROF\_ACC\_STATUS\_ANOTHER\_UTIL\_IN\_PROGRESS – another utility is running

- Reentrant : no
- ISR Callable : no

```
ixPerfProfAccXscalePmuTimeSampStop ( IxPerfProfAccXscalePmuEvtCnt * clkCount,
                                     IxPerfProfAccXscalePmuSamplePcProfile * timeProfile
                                     )
```

Stops the time based sampling.

**Parameters:**

- |   |  |
|---|--|
| <i>IxPerfProfAccXscalePmuEvtCnt [out]</i>                       | – pointer to the struct containing the final clock count and its overflow  |
| <i>*clkCount</i>  | It is the user's responsibility to allocate the memory for this pointer  |
| <i>IxPerfProfAccXscalePmuSamplePcProfile [out] *timeProfile</i> | – pointer to the array of profiles for each program counter value; user should set the size of the array to IX_PERFPROF_ACC_XSCALE_PMU_MAX_PROFILE_SAMPLES |
|   | It is the user's responsibility to allocate the memory for this pointer  |

This API stops the time based sampling. The results are stored in the pointers allocated by the user. It can only be called once ixPerfProfAccXscalePmuTimeSampStart has been called.

**Returns:**

- ◇ IX\_PERFPROF\_ACC\_STATUS\_SUCCESS if time based sampling is stopped successfully
- ◇ IX\_PERFPROF\_ACC\_STATUS\_XSCALE\_PMU\_START\_NOT\_CALLED if ixPerfProfAccXscalePmuTimeSampStart not called first

- Reentrant : no
- ISR Callable : no

# IXP425 Queue Manager (IxQMgr) API

The public API for the IXP425 QMgr component.

## Data Structures

struct **IxQMgrQInlinedReadWriteInfo**  
*Internal structure to facilitate inlining functions in IxQMgr.h.*

## Defines

#define **IX\_QMGR\_SAVED\_COMPONENT\_NAME**  
*Because this file contains inline functions which will be compiled into other components, we need to ensure that the IX\_COMPONENT\_NAME define is set to ix\_qmgr while this code is being compiled.*

#define **IX\_COMPONENT\_NAME**  
#define **IX\_QMGR\_INLINE**  
*Inline definition, for inlining of Queue Access functions on API.*

#define **IX\_QMGR\_MAX\_NUM\_QUEUES**  
*Number of queues supported by the AQM.*

#define **IX\_QMGR\_MIN\_QID**  
*Minimum queue identifier.*

#define **IX\_QMGR\_MAX\_QID**  
*Maximum queue identifier.*

#define **IX\_QMGR\_MIN\_QUEUEPP\_QID**  
*Minimum queue identifier for reduced functionality queues.*

#define **IX\_QMGR\_MAX\_QNAME\_LEN**  
*Maximum queue name length.*

#define **IX\_QMGR\_WARNING**  
*Warning return code.*

#define **IX\_QMGR\_PARAMETER\_ERROR**  
*Parameter error return code (NULL pointer etc..).*

#define **IX\_QMGR\_INVALID\_Q\_ENTRY\_SIZE**  
*Invalid entry size return code.*

#define **IX\_QMGR\_INVALID\_Q\_ID**  
*Invalid queue identifier return code.*

```

#define IX_QMGR_INVALID_CB_ID
    Invalid callback identifier return code.

#define IX_QMGR_CB_ALREADY_SET
    Callback set error return code.

#define IX_QMGR_NO_AVAILABLE_SRAM
    Sram consumed return code.

#define IX_QMGR_INVALID_INT_SOURCE_ID
    Invalid queue interrupt source identifier return code.

#define IX_QMGR_INVALID_QSIZE
    Invalid queue size error code.

#define IX_QMGR_INVALID_Q_WM
    Invalid queue watermark return code.

#define IX_QMGR_Q_NOT_CONFIGURED
    Queue not configured return code.

#define IX_QMGR_Q_ALREADY_CONFIGURED
    Queue already configured return code.

#define IX_QMGR_Q_UNDERFLOW
    Underflow return code.

#define IX_QMGR_Q_OVERFLOW
    Overflow return code.

#define IX_QMGR_Q_INVALID_PRIORITY
    Invalid priority return code.

#define IX_QMGR_ENTRY_INDEX_OUT_OF_BOUNDS
    Entry index out of bounds return code.

#define ixQMgrDispatcherLoopRun
    Map old function name ixQMgrDispatcherLoopRun() to
    ixQMgrDispatcherLoopRunA0().

```

## Typedefs

```

typedef UINT32 IxQMgrQStatus
    Queue status.

typedef unsigned IxQMgrCallbackId
    Uniquely identifies a callback function.

typedef void(* IxQMgrCallback )(IxQMgrQId qId, IxQMgrCallbackId cbId)

```

*QMgr notification callback type.*

```
typedef void(* IxQMgrDispatcherFuncPtr )(IxQMgrDispatchGroup group)  
QMgr Dispatcher Loop function pointer.
```

## Enumerations

```
enum IxQMgrQId {  
    IX_QMGR_QUEUE_0,  
    IX_QMGR_QUEUE_1,  
    IX_QMGR_QUEUE_2,  
    IX_QMGR_QUEUE_3,  
    IX_QMGR_QUEUE_4,  
    IX_QMGR_QUEUE_5,  
    IX_QMGR_QUEUE_6,  
    IX_QMGR_QUEUE_7,  
    IX_QMGR_QUEUE_8,  
    IX_QMGR_QUEUE_9,  
    IX_QMGR_QUEUE_10,  
    IX_QMGR_QUEUE_11,  
    IX_QMGR_QUEUE_12,  
    IX_QMGR_QUEUE_13,  
    IX_QMGR_QUEUE_14,  
    IX_QMGR_QUEUE_15,  
    IX_QMGR_QUEUE_16,  
    IX_QMGR_QUEUE_17,  
    IX_QMGR_QUEUE_18,  
    IX_QMGR_QUEUE_19,  
    IX_QMGR_QUEUE_20,  
    IX_QMGR_QUEUE_21,  
    IX_QMGR_QUEUE_22,  
    IX_QMGR_QUEUE_23,  
    IX_QMGR_QUEUE_24,  
    IX_QMGR_QUEUE_25,  
    IX_QMGR_QUEUE_26,  
    IX_QMGR_QUEUE_27,  
    IX_QMGR_QUEUE_28,  
    IX_QMGR_QUEUE_29,  
    IX_QMGR_QUEUE_30,  
    IX_QMGR_QUEUE_31,  
    IX_QMGR_QUEUE_32,  
    IX_QMGR_QUEUE_33,  
    IX_QMGR_QUEUE_34,  
    IX_QMGR_QUEUE_35,  
    IX_QMGR_QUEUE_36,  
    IX_QMGR_QUEUE_37,  
    IX_QMGR_QUEUE_38,  
    IX_QMGR_QUEUE_39,  
    IX_QMGR_QUEUE_40,
```

```

IX_QMGR_QUEUE_41,
IX_QMGR_QUEUE_42,
IX_QMGR_QUEUE_43,
IX_QMGR_QUEUE_44,
IX_QMGR_QUEUE_45,
IX_QMGR_QUEUE_46,
IX_QMGR_QUEUE_47,
IX_QMGR_QUEUE_48,
IX_QMGR_QUEUE_49,
IX_QMGR_QUEUE_50,
IX_QMGR_QUEUE_51,
IX_QMGR_QUEUE_52,
IX_QMGR_QUEUE_53,
IX_QMGR_QUEUE_54,
IX_QMGR_QUEUE_55,
IX_QMGR_QUEUE_56,
IX_QMGR_QUEUE_57,
IX_QMGR_QUEUE_58,
IX_QMGR_QUEUE_59,
IX_QMGR_QUEUE_60,
IX_QMGR_QUEUE_61,
IX_QMGR_QUEUE_62,
IX_QMGR_QUEUE_63,
IX_QMGR_QUEUE_INVALID
}

```

*Generic identifiers for AQM queues.*

```

enum IxQMGrQStatusMask {
    IX_QMGR_Q_STATUS_E_BIT_MASK,
    IX_QMGR_Q_STATUS_NE_BIT_MASK,
    IX_QMGR_Q_STATUS_NF_BIT_MASK,
    IX_QMGR_Q_STATUS_F_BIT_MASK,
    IX_QMGR_Q_STATUS_UF_BIT_MASK,
    IX_QMGR_Q_STATUS_OF_BIT_MASK
}

```

*Queue status mask.*

```

enum IxQMGrSourceId {
    IX_QMGR_Q_SOURCE_ID_E,
    IX_QMGR_Q_SOURCE_ID_NE,
    IX_QMGR_Q_SOURCE_ID_NF,
    IX_QMGR_Q_SOURCE_ID_F,
    IX_QMGR_Q_SOURCE_ID_NOT_E,
    IX_QMGR_Q_SOURCE_ID_NOT_NE,
    IX_QMGR_Q_SOURCE_ID_NOT_NF,
    IX_QMGR_Q_SOURCE_ID_NOT_F
}

```

*Queue interrupt source select.*

```

enum IxQMGrQEntrySizeInWords {
    IX_QMGR_Q_ENTRY_SIZE1,

```

```

IX_QMGR_Q_ENTRY_SIZE2,
IX_QMGR_Q_ENTRY_SIZE4
}

```

*QMgr queue entry sizes.*

```

enum IxQMgrQSizeInWords {
    IX_QMGR_Q_SIZE16,
    IX_QMGR_Q_SIZE32,
    IX_QMGR_Q_SIZE64,
    IX_QMGR_Q_SIZE128,
    IX_QMGR_Q_SIZE_INVALID
}

```

*QMgr queue sizes.*

```

enum IxQMgrWMLevel {
    IX_QMGR_Q_WM_LEVEL0,
    IX_QMGR_Q_WM_LEVEL1,
    IX_QMGR_Q_WM_LEVEL2,
    IX_QMGR_Q_WM_LEVEL4,
    IX_QMGR_Q_WM_LEVEL8,
    IX_QMGR_Q_WM_LEVEL16,
    IX_QMGR_Q_WM_LEVEL32,
    IX_QMGR_Q_WM_LEVEL64
}

```

*QMgr watermark levels.*

```

enum IxQMgrDispatchGroup {
    IX_QMGR_QUELOW_GROUP,
    IX_QMGR_QUEUPP_GROUP
}

```

*QMgr dispatch group select identifiers.*

```

enum IxQMgrPriority {
    IX_QMGR_Q_PRIORITY_0,
    IX_QMGR_Q_PRIORITY_1,
    IX_QMGR_Q_PRIORITY_2,
    IX_QMGR_Q_PRIORITY_INVALID
}

```

*Dispatcher priority levels.*

## Functions

```

PUBLIC IX_STATUS ixQMgrInit (void)
    Initialise the QMgr.

```

```

PUBLIC IX_STATUS ixQMgrUnload (void)
    Uninitialise the QMgr.

```

```

PUBLIC void ixQMgrShow (void)

```

*Describe queue configuration and statistics for active queues.*

**PUBLIC IX\_STATUS ixQMgrQShow (IxQMgrQId qId)**

*Display a queue configuration and statistics for a queue.*

**PUBLIC IX\_STATUS ixQMgrQConfig (char \*qName, IxQMgrQId qId,  
IxQMgrQSizeInWords qSizeInWords, IxQMgrQEntrySizeInWords  
qEntrySizeInWords)**

*Configure an AQM queue.*

**PUBLIC IX\_STATUS ixQMgrQSizeInEntriesGet (IxQMgrQId qId, unsigned  
\*qSizeInEntries)**

*Return the size of a queue in entries.*

**PUBLIC IX\_STATUS ixQMgrWatermarkSet (IxQMgrQId qId, IxQMgrWMLevel ne,  
IxQMgrWMLevel nf)**

*Set the Nearly Empty and Nearly Full Watermarks for a queue.*

**PUBLIC IX\_STATUS ixQMgrAvailableSramAddressGet (UINT32 \*address, unsigned  
\*sizeOfFreeSram)**

*Return the address of available AQM SRAM.*

**PUBLIC IX\_STATUS ixQMgrQReadWithChecks (IxQMgrQId qId, UINT32 \*entry)**

*Read an entry from a queue.*

**IX\_STATUS ixQMgrQReadMWordsMinus1 (IxQMgrQId qId, UINT32 \*entry)**

*This function reads the remaining of the q entry for queues configured  
with many words. (the first word of the entry is already read in the  
inlined function and the entry pointer already incremented).*

**IX\_QMGR\_INLINE PUBLIC**

**IX\_STATUS ixQMgrQRead (IxQMgrQId qId, UINT32 \*entryPtr)**

*Fast read of an entry from a queue.*

**IX\_QMGR\_INLINE PUBLIC ixQMgrQBurstRead (IxQMgrQId qId, UINT32 numEntries, UINT32**

**IX\_STATUS \*entries)**

*Read a number of entries from an AQM queue.*

**IX\_STATUS ixQMgrQPeek (IxQMgrQId qId, unsigned int entryIndex, UINT32  
\*entry)**

*Read an entry from a queue without moving the read pointer.*

**PUBLIC IX\_STATUS ixQMgrQWriteWithChecks (IxQMgrQId qId, UINT32 \*entry)**

*Write an entry to an AQM queue.*

**IX\_QMGR\_INLINE PUBLIC**

**IX\_STATUS ixQMgrQWrite (IxQMgrQId qId, UINT32 \*entry)**

*Fast write of an entry to a queue.*

**IX\_QMGR\_INLINE PUBLIC ixQMgrQBurstWrite (IxQMgrQId qId, unsigned numEntries, UINT32**

**IX\_STATUS \*entries)**



*Write a number of entries to an AQM queue.*

**IX\_STATUS ixQMgrQPoke (IxQMgrQId qId, unsigned int entryIndex, UINT32 \*entry)**

*Write an entry to a queue without moving the write pointer.*

**PUBLIC IX\_STATUS ixQMgrQNumEntriesGet (IxQMgrQId qId, unsigned \*numEntries)**  
*Get a snapshot of the number of entries in a queue.*

**PUBLIC IX\_STATUS ixQMgrQStatusGetWithChecks (IxQMgrQId qId, IxQMgrQStatus \*qStatus)**  
*Get a queues status.*

**IX\_QMGR\_INLINE PUBLIC**

**IX\_STATUS ixQMgrQStatusGet (IxQMgrQId qId, IxQMgrQStatus \*qStatus)**  
*Fast get of a queue's status.*

**PUBLIC IX\_STATUS ixQMgrDispatcherPrioritySet (IxQMgrQId qId, IxQMgrPriority priority)**  
*Set the dispatch priority of a queue.*

**PUBLIC IX\_STATUS ixQMgrNotificationEnable (IxQMgrQId qId, IxQMgrSourceId sourceId)**  
*Enable notification on a queue for a specified queue source flag.*

**PUBLIC IX\_STATUS ixQMgrNotificationDisable (IxQMgrQId qId)**  
*Disable notifications on a queue.*

**PUBLIC void ixQMgrDispatcherLoopRunA0 (IxQMgrDispatchGroup group)**  
*Run the callback dispatcher.*

**PUBLIC void ixQMgrDispatcherLoopRunB0 (IxQMgrDispatchGroup group)**  
*Run the callback dispatcher.*

**PUBLIC IX\_STATUS ixQMgrNotificationCallbackSet (IxQMgrQId qId, IxQMgrCallback callback, IxQMgrCallbackId callbackId)**  
*Set the notification callback for a queue.*

**PUBLIC void ixQMgrDispatcherLoopGet (IxQMgrDispatcherFuncPtr \*qDispatcherFuncPtr)**  
*Get QMgr DispatcherLoopRun for respective silicon device.*

**PUBLIC void ixQMgrStickyInterruptRegEnable (void)**  
*Enable AQM's sticky interrupt register behaviour only available on B0 Silicon.*

## Variables

**IxQMgrQInlinedReadWriteInfo** **ixQMgrQInlinedReadWriteInfo** []  
    UINT32 **ixQMgrAqmIfQueLowStatRegAddr** []  
    UINT32 **ixQMgrAqmIfQueLowStatBitsOffset** []  
    UINT32 **ixQMgrAqmIfQueLowStatBitsMask**  
    UINT32 **ixQMgrAqmIfQueUppStat0RegAddr**  
    UINT32 **ixQMgrAqmIfQueUppStat1RegAddr**  
    UINT32 **ixQMgrAqmIfQueUppStat0BitMask** []  
    UINT32 **ixQMgrAqmIfQueUppStat1BitMask** []

---

## Detailed Description

The public API for the IXP425 QMgr component.

IxQMgr is a low level interface to the AHB Queue Manager

---

## Define Documentation

```
#define IX_QMGR_CB_ALREADY_SET
```

Callback set error return code.

The specified callback has already been for this queue

Definition at line **273** of file **IxQMgr.h**.

```
#define IX_QMGR_ENTRY_INDEX_OUT_OF_BOUNDS
```

Entry index out of bounds return code.

Entry index is greater than number of entries in queue.

Definition at line **400** of file **IxQMgr.h**.

```
#define IX_QMGR_INLINE
```

Inline definition, for inlining of Queue Access functions on API.

Please read the header information in this file for more details on the use of function inlining in this component.

Definition at line **125** of file **IxQMgr.h**.

```
#define IX_QMGR_INVALID_CB_ID
```

Invalid callback identifier return code.

Invalid callback id

Definition at line **260** of file **IxQMgr.h**.

```
#define IX_QMGR_INVALID_INT_SOURCE_ID
```

Invalid queue interrupt source identifier return code.

Invalid queue interrupt source given for notification enable

Definition at line **298** of file **IxQMgr.h**.

```
#define IX_QMGR_INVALID_Q_ENTRY_SIZE
```

Invalid entry size return code.

Invalid queue entry size for a queue read/write

Definition at line **235** of file **IxQMgr.h**.

```
#define IX_QMGR_INVALID_Q_ID
```

Invalid queue identifier return code.

Invalid queue id, not in range 0–63

Definition at line **248** of file **IxQMgr.h**.

```
#define IX_QMGR_INVALID_Q_WM
```

Invalid queue watermark return code.

Invalid queue watermark given for watermark set

Definition at line **324** of file **IxQMgr.h**.

```
#define IX_QMGR_INVALID_QSIZE
```

Invalid queue size error code.

Invalid queue size not one of 16,32, 64, 128

Definition at line **312** of file **IxQMgr.h**.

```
#define IX_QMGR_MAX_NUM_QUEUES
```

Number of queues supported by the AQM.

This constant is used to indicate the number of AQM queues

Definition at line **142** of file **IxQMgr.h**.

```
#define IX_QMGR_MAX_QID
```

Maximum queue identifier.

This constant is used to indicate the largest queue identifier

Definition at line **168** of file **IxQMgr.h**.

```
#define IX_QMGR_MAX_QNAME_LEN
```

Maximum queue name length.

This constant is used to indicate the maximum null terminated string length (excluding ") for a queue name

Definition at line **196** of file **IxQMgr.h**.

```
#define IX_QMGR_MIN_QID
```

Minimum queue identifier.

This constant is used to indicate the smallest queue identifier

Definition at line **155** of file **IxQMgr.h**.

```
#define IX_QMGR_MIN_QUEUPP_QID
```

Minimum queue identifier for reduced functionality queues.

This constant is used to indicate Minimum queue identifier for reduced functionality queues

Definition at line **182** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstRead()**, **ixQMgrQBurstWrite()**, **ixQMgrQRead()**, **ixQMgrQStatusGet()**, and **ixQMgrQWrite()**.

```
#define IX_QMGR_NO_AVAILABLE_SRAM
```

Sram consumed return code.

All AQM Sram is consumed by queue configuration

Definition at line **286** of file **IxQMgr.h**.

```
#define IX_QMGR_PARAMETER_ERROR
```

Parameter error return code (NULL pointer etc..).

parameter error out of range/invalid

Definition at line **222** of file **IxQMgr.h**.

```
#define IX_QMGR_Q_ALREADY_CONFIGURED
```

Queue already configured return code.

Returned to client to indicate that a queue has already been configured

Definition at line **350** of file **IxQMgr.h**.

```
#define IX_QMGR_Q_INVALID_PRIORITY
```

Invalid priority return code.

Invalid priority, not one of 0,1,2

Definition at line **388** of file **IxQMgr.h**.

```
#define IX_QMGR_Q_NOT_CONFIGURED
```

Queue not configured return code.

Returned to the client when a function has been called on an unconfigured queue

Definition at line **338** of file **IxQMgr.h**.

```
#define IX_QMGR_Q_OVERFLOW
```

Overflow return code.

Overflow on a queue write has occurred

Definition at line **376** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstWrite()**, and **ixQMgrQWrite()**.

```
#define IX_QMGR_Q_UNDERFLOW
```

Underflow return code.

Underflow on a queue read has occurred

Definition at line **363** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstRead()**, and **ixQMgrQRead()**.

```
#define IX_QMGR_SAVED_COMPONENT_NAME
```

Because this file contains inline functions which will be compiled into other components, we need to ensure that the **IX\_COMPONENT\_NAME** define is set to **ix\_qmgr** while this code is being compiled.

This will ensure that the correct implementation is provided for the memory access macros **IX\_OSSERV\_READ\_LONG** and **IX\_OSSERV\_WRITE\_LONG** which are used in this file. This must be done before including "IxOsServicesMemAccess.h"

Definition at line **93** of file **IxQMgr.h**.

```
#define IX_QMGR_WARNING
```

Warning return code.

Execution complete, but there is a special case to handle

Definition at line **209** of file **IxQMgr.h**.

```
#define ixQMgrDispatcherLoopRun
```

Map old function name **ixQMgrDispatcherLoopRun()** to **ixQMgrDispatcherLoopRunA0()**.

Definition at line **412** of file **IxQMgr.h**.

---

# Typedef Documentation

## IxQMgrCallback

QMgr notification callback type.

This defines the interface to all client callback functions.

**Parameters:**

*IxQMgrQId*                qId(in) – the queue identifier  
*IxQMgrCallbackId*    cbId(in) – the callback  
   identifier

Definition at line **699** of file **IxQMgr.h**.

## IxQMgrCallbackId

Uniquely identifies a callback function.

A unique callback identifier associated with each callback registered by clients.

Definition at line **687** of file **IxQMgr.h**.

## IxQMgrDispatcherFuncPtr

QMgr Dispatcher Loop function pointer.

This defines the interface for QMgr Dispatcher functions.

**Parameters:**

*IxQMgrDispatchGroup* group(in) – the group of the queue of which the dispatcher  
   will run

Definition at line **714** of file **IxQMgr.h**.

## IxQMgrQStatus

Queue status.

A queues status is defined by its relative fullness or relative emptiness. Each of the queues 0–31 have Nearly Empty, Nearly Full, Empty, Full, Underflow and Overflow status flags. Queues 32–63 have just Nearly Empty and Full status flags. The flags bit positions are outlined below:

OF – bit–5  
UF – bit–4

F – bit–3  
NF – bit–2  
NE – bit–1  
E – bit–0

Definition at line **523** of file **IxQMgr.h**.

---

## Enumeration Type Documentation

### enum IxQMgrDispatchGroup

QMgr dispatch group select identifiers.

This enum defines the groups over which the dispatcher will process when called. One of the enum values must be used as a input to **ixQMgrDispatcherLoopRun()**.

**Enumeration values:**

**IX\_QMGR\_QUELOW\_GROUP** Queues 0–31.

**IX\_QMGR\_QUEUPP\_GROUP** Queues 32–63.

Definition at line **651** of file **IxQMgr.h**.

### enum IxQMgrPriority

Dispatcher priority levels.

This enum defines the different queue dispatch priority levels. The lowest priority number (0) is the highest priority level.

**Enumeration values:**

**IX\_QMGR\_Q\_PRIORITY\_0** Priority level 0.

**IX\_QMGR\_Q\_PRIORITY\_1** Priority level 1.

**IX\_QMGR\_Q\_PRIORITY\_2** Priority level 2.

**IX\_QMGR\_Q\_PRIORITY\_INVALID** Invalid Priority level.

Definition at line **668** of file **IxQMgr.h**.

### enum IxQMgrQEntrySizeInWords

QMgr queue entry sizes.

The entry size of a queue specifies the size of a queues entry in words.

**Enumeration values:**

**IX\_QMGR\_Q\_ENTRY\_SIZE1** 1 word entry



*IX\_QMGR\_Q\_ENTRY\_SIZE2* 2 word entry  
*IX\_QMGR\_Q\_ENTRY\_SIZE4* 4 word entry

Definition at line **581** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstRead()**, and **ixQMgrQBurstWrite()**.

#### enum IxQMgrQId

Generic identifiers for AQM queues.

These enum values are used in the AQM queue config header file. This enum defines generic identifiers for the queues in that the application of the queue is not specified. The connection between queue number and queue application is done in the AQM queue config header file. Clients of this component should NOT use these values to identify queues, the #defines in the AQM queue configuration header should be used.

Definition at line **433** of file **IxQMgr.h**.

#### enum IxQMgrQSizeInWords

QMgr queue sizes.

These values define the allowed queue sizes for AQM queue. The sizes are specified in words.

##### **Enumeration values:**

<i>IX_QMGR_Q_SIZE16</i>	16 word buffer
<i>IX_QMGR_Q_SIZE32</i>	32 word buffer
<i>IX_QMGR_Q_SIZE64</i>	64 word buffer
<i>IX_QMGR_Q_SIZE128</i>	128 word buffer
<i>IX_QMGR_Q_SIZE_INVALID</i>	Insure that this is greater than largest queue size supported by the hardware.

Definition at line **599** of file **IxQMgr.h**.

#### enum IxQMgrQStatusMask

Queue status mask.

Masks for extracting the individual status flags from the IxQMgrStatus word.

Definition at line **536** of file **IxQMgr.h**.

#### enum IxQMgrSourceId

Queue interrupt source select.

This enum defines the different source conditions on a queue that result in an interrupt being fired by the AQM. Interrupt source is configurable for queues 0–31 only. The interrupt source for queues 32–63 is hardwired to the NE(Nearly Empty) status flag.

**Enumeration values:**

<i>IX_QMGR_Q_SOURCE_ID_E</i>	Queue Empty due to last read.
<i>IX_QMGR_Q_SOURCE_ID_NE</i>	Queue Nearly Empty due to last read.
<i>IX_QMGR_Q_SOURCE_ID_NF</i>	Queue Nearly Full due to last write.
<i>IX_QMGR_Q_SOURCE_ID_F</i>	Queue Full due to last write.
<i>IX_QMGR_Q_SOURCE_ID_NOT_E</i>	Queue Not Empty due to last write.
<i>IX_QMGR_Q_SOURCE_ID_NOT_NE</i>	Queue Not Nearly Empty due to last write.
<i>IX_QMGR_Q_SOURCE_ID_NOT_NF</i>	Queue Not Nearly Full due to last read.
<i>IX_QMGR_Q_SOURCE_ID_NOT_F</i>	Queue Not Full due to last read.

Definition at line **559** of file **IxQMgr.h**.

```
enum IxQMgrWMLevel
```

QMgr watermark levels.

These values define the valid watermark levels(in ENTRIES) for queues. Each queue 0–63 have configurable Nearly full and Nearly empty watermarks. For queues 32–63 the Nearly full watermark has NO EFFECT. If the Nearly full watermark is set to *IX\_QMGR\_Q\_WM\_LEVEL16* this means that the nearly full flag will be set by the hardware when there are  $\geq 16$  empty entries in the specified queue. If the Nearly empty watermark is set to *IX\_QMGR\_Q\_WM\_LEVEL16* this means that the Nearly empty flag will be set by the hardware when there are  $\leq 16$  full entries in the specified queue.

**Enumeration values:**

<i>IX_QMGR_Q_WM_LEVEL0</i>	0 entry watermark
<i>IX_QMGR_Q_WM_LEVEL1</i>	1 entry watermark
<i>IX_QMGR_Q_WM_LEVEL2</i>	2 entry watermark
<i>IX_QMGR_Q_WM_LEVEL4</i>	4 entry watermark
<i>IX_QMGR_Q_WM_LEVEL8</i>	8 entry watermark
<i>IX_QMGR_Q_WM_LEVEL16</i>	16 entry watermark
<i>IX_QMGR_Q_WM_LEVEL32</i>	32 entry watermark
<i>IX_QMGR_Q_WM_LEVEL64</i>	64 entry watermark

Definition at line **627** of file **IxQMgr.h**.

---

## Function Documentation

```
ixQMgrAvailableSramAddressGet ( UINT32 * address,  
                                unsigned * sizeOfFreeSram  
                                )
```

Return the address of available AQM SRAM.

This function returns the starting address in AQM SRAM not used by the current queue configuration and should only be called after all queues have been configured. Calling this function before all queues have been configured will return the currently available SRAM. A call to configure another queue will use some of the available SRAM. The amount of SRAM available is specified in `sizeofFreeSram`. The address is the address of the bottom of available SRAM. Available SRAM extends from address from address to `address + sizeofFreeSram`.

**Parameters:**

<i>UINT32(out)</i>	**address – the address of the available SRAM, NULL if none available.
<i>unsigned(out)</i>	*sizeofFreeSram – the size in words of available SRAM

**Returns:**

- ◇ IX\_SUCCESS, there is available SRAM and is pointed to by address
- ◇ IX\_QMGR\_PARAMETER\_ERROR, invalid parameter(s)
- ◇ IX\_QMGR\_NO\_AVAILABLE\_SRAM, all AQM SRAM is consumed by the queue configuration.

```
ixQMgrDispatcherLoopGet ( IxQMgrDispatcherFuncPtr * qDispatcherFuncPtr )
```

Get QMgr DispatcherLoopRun for respective silicon device.

This function get the function pointer for **ixQMgrDispatcherLoopRunA0()** for A0 or **ixQMgrDispatcherLoopRunB0()** for B0 Silicon.

**Parameters:**

*IxQMgrDispatcherFuncPtr(out)* \*qDispatcherFuncPtr – the function pointer of QMgr Dispatcher

```
ixQMgrDispatcherLoopRunA0 ( IxQMgrDispatchGroup group )
```

Run the callback dispatcher.

The function runs the dispatcher for a group of queues. Callbacks are made for interrupts that have occurred on queues within the group that have registered callbacks. The order in which queues are serviced depends on the queue priorities set by the client. This function may be called from interrupt or task context.

This function is not re-entrant.

**Parameters:**

*IxQMgrDispatchGroup(in)* *group* – the group of queues over which the dispatcher will run

**Returns:**

- ◇ void

**Note:**

This function may be called from interrupt or task context. However, for optimal performance the choice of context depends also on the operating system used.

For optimal performance with the WindRiver VxWorks OS the recommendation is to use a polling task to call the Dispatcher, for the following reasons:

- The system cannot handle more than 42000 interrupts per second. As a result, the system may crash at 25% of the maximum traffic rate for Ethernet traffic.
- Using a polling task to call the Dispatcher works fine.

For optimal performance with the Linux OS the recommendation is to use interrupts to call the Dispatcher, for the following reasons:

- A task loop cannot run more than 100 times per second (HZ). As an example, if an Ethernet port uses a queue size of 128 entries, the maximum achievable traffic rate is 12800 packets per second. (the system does not crash, but this is 8% of wire speed)
- Using queue interrupts to call the Dispatcher works fine.

**ixQMgrDispatcherLoopRunB0 ( *IxQMgrDispatchGroup* group )**

Run the callback dispatcher.

The enhanced version of ixQMgrDispatcherLoopRun () for B0 silicon. The function runs the dispatcher for a group of queues. Callbacks are made for interrupts that have occurred on queues within the group that have registered callbacks. The order in which queues are serviced depends on the queue priorities set by the client. This function may be called from interrupt or task context.

This function is not re-entrant.

**Parameters:**

*IxQMgrDispatchGroup*(in) group – the group of queues over which the dispatcher will run

**Returns:**

◇ void

**Note:**

Compared with **ixQMgrDispatcherLoopRunA0()**, **ixQMgrDispatcherLoopRunB0()** is more efficient and takes advantage of QMgr optimization in B0 silicon .

This function may be called from interrupt or task context. However, for optimal performance the choice of context depends also on the operating system used.

For optimal performance with the WindRiver VxWorks OS the recommendation is to use a polling task to call the Dispatcher, for the following reasons:

- The system cannot handle more than 42000 interrupts per second. As a result, the system may crash at 25% of the maximum traffic rate for Ethernet traffic.
- Using a polling task to call the Dispatcher works fine.

For optimal performance with the Linux OS the recommendation is to use interrupts to call the Dispatcher, for the following reasons:

- A task loop cannot run more than 100 times per second (HZ). As an example, if an Ethernet port uses a queue size of 128 entries, the maximum achievable traffic rate is 12800 packets per second. (the system does not crash, but this is 8% of wire speed)
- Using queue interrupts to call the Dispatcher works fine.

```
ixQMgrDispatcherPrioritySet ( IxQMgrQId      qId,  
                             IxQMgrPriority priority  
                             )
```

Set the dispatch priority of a queue.

This function is called to set the dispatch priority of queue. The effect of this function is to add a priority change request to a queue. This queue is serviced by *ixQMgrDispatcherLoopRun*.

This function is re-entrant. and can be used from an interrupt context

**Parameters:**

*IxQMgrQId(in)*      *qId* – the queue identifier  
*IxQMgrPriority(in)* *priority* – the new queue dispatch priority

**Returns:**

- ◇ IX\_SUCCESS, priority change request is queued
- ◇ IX\_QMGR\_Q\_NOT\_CONFIGURED, the specified *qId* has not been configured
- ◇ IX\_QMGR\_Q\_INVALID\_PRIORITY, specified priority is invalid

```
ixQMgrInit ( void )
```

Initialise the QMgr.

This function must be called before any other QMgr function. It sets up internal data structures.

**Returns:**

- ◇ IX\_SUCCESS, the IxQMgr successfully initialised
- ◇ IX\_FAIL, failed to initialize the Qmgr

```
ixQMgrNotificationCallbackSet ( IxQMgrQId      qId,  
                                IxQMgrCallback callback,  
                                IxQMgrCallbackId callbackId  
                                )
```

Set the notification callback for a queue.

This function sets the callback for the specified queue. This callback will be called by the dispatcher, and may be called in the context of an interrupt. If callback has a value of NULL the previously registered callback, if one exists will be unregistered.

**Parameters:**

*IxQMgrQId(in)*                qId – the queue identifier  
*IxQMgrCallback(in)*        callback – the callback registered for this queue  
*IxQMgrCallbackId(in)*    callbackid – the callback identifier

**Returns:**

- ◊ IX\_SUCCESS, the callback for the specified queue has been set
- ◊ IX\_QMGR\_Q\_NOT\_CONFIGURED, the specified qId has not been configured

```
ixQMgrNotificationDisable ( IxQMgrQId qId )
```

Disable notifications on a queue.

This function is called to disable notifications on a specified queue.

This function is re-entrant. and can be used from an interrupt context

**Parameters:**

*IxQMgrQId(in)*    qId – the queue identifier

**Returns:**

- ◊ IX\_SUCCESS, the interrupt has been disabled
- ◊ IX\_QMGR\_Q\_NOT\_CONFIGURED, the specified qId has not been configured

```
ixQMgrNotificationEnable ( IxQMgrQId        qId,  
                          IxQMgrSourceId sourceId  
                          )
```

Enable notification on a queue for a specified queue source flag.

This function is called by a client of the QMgr to enable notifications on a specified condition. If the condition for the notification is set after the client has called this function but before the function has enabled the interrupt source, then the notification will not occur. For queues 32–63 the notification source is fixed to the NE(Nearly Empty) flag and cannot be changed so the sourceId parameter is ignored for these queues. The status register is read before the notification is enabled and is read again after the notification has been enabled, if they differ then the warning status is returned.

This function is re-entrant. and can be used from an interrupt context

**Parameters:**

*IxQMgrQId(in)*            qId – the queue identifier  
*IxQMgrSourceId(in)*    sourceId – the interrupt src condition identifier

**Returns:**

- ◇ IX\_SUCCESS, the interrupt has been enabled for the specified source
- ◇ IX\_QMGR\_Q\_NOT\_CONFIGURED, the specified qId has not been configured
- ◇ IX\_QMGR\_INVALID\_INT\_SOURCE\_ID, interrupt source invalid for this queue
- ◇ IX\_QMGR\_WARNING, the status register may not be consistent

```
IX_QMGR_INLINE PUBLIC IX_STATUS ixQMgrQBurstRead ( IxQMgrQId qId,
                                                    UINT32      numEntries,
                                                    UINT32 *    entries
                                                    )
```

Read a number of entries from an AQM queue.

This function will burst read a number of entries from the specified queue. The entry size of queue is auto-detected. The function will attempt to read as many entries as specified by the numEntries parameter and will return an UNDERFLOW if any one of the individual entry reads fail.

**Warning:**

IX\_QMGR\_Q\_UNDERFLOW is only returned for queues 0–31 as queues 32–63 do not have an underflow status maintained, hence there is a potential for silent failure here. This function must be used with caution.

**Note:**

This function is intended for fast draining of queues, so to make it as efficient as possible, it has the following features:

- ◇ This function is inlined, to reduce unnecessary function call overhead.
- ◇ It does not perform any parameter checks, or update any statistics.
- ◇ It does not check that the queue specified by qId has been configured.
- ◇ It does not check that the queue has the number of full entries that have been specified to be read. It will read until it finds a NULL entry or until the number of specified entries have been read. It always checks for underflow after all the reads have been performed. Therefore, the client should ensure before calling this function that there are enough entries in the queue to read. **ixQMgrQNumEntriesGet()** will provide the number of full entries in a queue. **ixQMgrQRead()** or **ixQMgrQReadWithChecks()**, which only reads a single queue entry per call, should be used instead if the user requires checks for UNDERFLOW after each entry read.

**Parameters:**

*IxQMgrQId(in)*            qId – the queue identifier.  
*unsigned(in)*            numEntries – the number of entries to read. This number should be greater than 0  
*UINT32(out)*            \*entries – the word(s) read.

**Returns:**

- ◊ `IX_SUCCESS`, entries were successfully read.
- ◊ `IX_QMGR_Q_UNDERFLOW`, attempt to read from an empty queue

Definition at line **1219** of file **IxQMgr.h**.

References **`IX_QMGR_MIN_QUEUEPP_QID`**, **`IX_QMGR_Q_ENTRY_SIZE1`**, **`IX_QMGR_Q_UNDERFLOW`**, **`IX_SUCCESS`**, **`IxQMgrQEntrySizeInWords`**, **`IxQMgrQInlinedReadWriteInfo::qAccRegAddr`**, **`IxQMgrQInlinedReadWriteInfo::qEntrySizeInWords`**, **`IxQMgrQInlinedReadWriteInfo::qReadCount`**, **`IxQMgrQInlinedReadWriteInfo::qUflowStatBitMask`**, and **`IxQMgrQInlinedReadWriteInfo::qUOStatRegAddr`**.

```
IX_QMGR_INLINE PUBLIC IX_STATUS ixQMgrQBurstWrite ( IxQMgrQId qId,
                                                    unsigned      numEntries,
                                                    UINT32 *      entries
                                                    )
```

Write a number of entries to an AQM queue.

This function will burst write a number of entries to the specified queue. The entry size of queue is auto-detected. The function will attempt to write as many entries as specified by the `numEntries` parameter and will return an `OVERFLOW` if any one of the individual entry writes fail.

**Warning:**

`IX_QMGR_Q_OVERFLOW` is only returned for queues 0–31 as queues 32–63 do not have an overflow status maintained, hence there is a potential for silent failure here. This function must be used with caution.

**Note:**

This function is intended for fast population of queues, so to make it as efficient as possible, it has the following features:

- ◊ This function is inlined, to reduce unnecessary function call overhead.
- ◊ It does not perform any parameter checks, or update any statistics.
- ◊ It does not check that the queue specified by `qId` has been configured.
- ◊ It does not check that the queue has enough free space to hold the entries before writing, and only checks for overflow after all writes have been performed. Therefore, the client should ensure before calling this function that there is enough free space in the queue to hold the number of entries to be written. **`ixQMgrQWrite()`** or **`ixQMgrQWriteWithChecks()`**, which only writes a single queue entry per call, should be used instead if the user requires checks for `OVERFLOW` after each entry written.

**Parameters:**

*IxQMgrQId(in)* `qId` – the queue identifier.  
*unsigned(in)* `numEntries` – the number of entries to write.  
*UINT32(in)* `*entries` – the word(s) to write.



**Returns:**

- ◇ IX\_SUCCESS, value was successfully written.
- ◇ IX\_QMGR\_Q\_OVERFLOW, attempt to write to a full queue

Definition at line **1561** of file **IxQMgr.h**.

References **IX\_QMGR\_MIN\_QUEUEUPP\_QID**, **IX\_QMGR\_Q\_ENTRY\_SIZE1**, **IX\_QMGR\_Q\_OVERFLOW**, **IX\_SUCCESS**, **IxQMgrQEntrySizeInWords**, **IxQMgrQInlinedReadWriteInfo::qAccRegAddr**, **IxQMgrQInlinedReadWriteInfo::qEntrySizeInWords**, **IxQMgrQInlinedReadWriteInfo::qOflowStatBitMask**, **IxQMgrQInlinedReadWriteInfo::qSizeInEntries**, **IxQMgrQInlinedReadWriteInfo::qUOStatRegAddr**, and **IxQMgrQInlinedReadWriteInfo::qWriteCount**.

```

IxQMgrQConfig ( char *           qName,
                  IxQMgrQId       qId,
                  IxQMgrQSizeInWords qSizeInWords,
                  IxQMgrQEntrySizeInWords qEntrySizeInWords
                )

```

Configure an AQM queue.

This function is called by a client to setup a queue. The size and entrySize qId and qName(NULL pointer) are checked for valid values. This function must be called for each queue, before any queue accesses are made and after **ixQMgrInit()** has been called. qName is assumed to be a " terminated array of 16 charachters or less.

**Parameters:**

- |                                    |   |
|------------------------------------|---|
| <i>char(in)</i>                    | *qName – is the name provided by the client and is associated with a QId by the QMgr. |
| <i>IxQMgrQId(in)</i>               | qId – the qId of this queue   |
| <i>IxQMgrQSize(in)</i>             | qSizeInWords – the size of the queue can be one of 16,32 64, 128 words.               |
| <i>IxQMgrQEntrySizeInWords(in)</i> | qEntrySizeInWords – the size of a queue entry can be one of 1,2,4 words.              |

**Returns:**

- ◇ IX\_SUCCESS, a specified queue has been successfully configured.
- ◇ IX\_FAIL, IxQMgr has not been initialised.
- ◇ IX\_QMGR\_PARAMETER\_ERROR, invalid parameter(s).
- ◇ IX\_QMGR\_INVALID\_QSIZE, invalid queue size
- ◇ IX\_QMGR\_INVALID\_Q\_ID, invalid queue id

- ◇ IX\_QMGR\_INVALID\_Q\_ENTRY\_SIZE, invalid queue entry size
- ◇ IX\_QMGR\_Q\_ALREADY\_CONFIGURED, queue already configured

```
ixQMgrQNumEntriesGet ( IxQMgrQId qId,
                      unsigned *   numEntries
                      )
```

Get a snapshot of the number of entries in a queue.

This function gets the number of entries in a queue.

**Parameters:**

*IxQMgrQId(in)* qId – the queue identifier  
*unsigned(out)* \*numEntries – the number of entries in a queue

**Returns:**

- ◇ IX\_SUCCESS, got the number of entries for the queue
- ◇ IX\_QMGR\_PARAMETER\_ERROR, invalid paramter(s).
- ◇ IX\_QMGR\_Q\_NOT\_CONFIGURED, the specified qId has not been configured
- ◇ IX\_QMGR\_WARNING, could not determine num entries at this time

```
ixQMgrQPek ( IxQMgrQId qId,
             unsigned int  entryIndex,
             UINT32 *      entry
             )
```

Read an entry from a queue without moving the read pointer.

This function inspects an entry in a queue. The entry is inspected directly in AQM SRAM and is not read from queue access registers. The entry is NOT removed from the queue and the read/write pointers are unchanged. N.B: The queue should not be accessed when this function is called.

**Parameters:**

*IxQMgrQId(in)* qId – the queue identifier.  
*unsigned* int(in) entryIndex – index of entry in queue in the range [0].....[current number of entries in queue].  
*UINT32(out)* \*entry – pointer to the entry word(s).

**Returns:**

- ◇ IX\_SUCCESS, entry was successfully inspected.
- ◇ IX\_QMGR\_PARAMETER\_ERROR, invalid paramter(s).

- ◇ IX\_QMGR\_Q\_NOT\_CONFIGURED, queue not configured for this QId.
- ◇ IX\_QMGR\_ENTRY\_INDEX\_OUT\_OF\_BOUNDS, an entry does not exist at specified index.
- ◇ IX\_FAIL, failed to injected the queue entry.

```
ixQMgrQPoke ( IxQMgrQId qId,
               unsigned int  entryIndex,
               UINT32 *      entry
             )
```

Write an entry to a queue without moving the write pointer.

This function modifies an entry in a queue. The entry is modified directly in AQM SRAM and not using the queue access registers. The entry is NOT added to the queue and the read/write pointers are unchanged.

N.B: The queue should not be accessed when this function is called.

**Parameters:**

*IxQMgrQId(in)* qId – the queue identifier.

*unsigned int(in)* entryIndex – index of entry in queue in the range [0].....[current number of entries in queue].

*UINT32(in)* \*entry – pointer to the entry word(s).

**Returns:**

- ◇ IX\_SUCCESS, entry was successfully modified.
- ◇ IX\_QMGR\_PARAMETER\_ERROR, invalid paramter(s).
- ◇ IX\_QMGR\_Q\_NOT\_CONFIGURED, queue not configured for this QId.
- ◇ IX\_QMGR\_ENTRY\_INDEX\_OUT\_OF\_BOUNDS, an entry does not exist at specified index.
- ◇ IX\_FAIL, failed to modify the queue entry.

```
IX_QMGR_INLINE PUBLIC IX_STATUS ixQMgrQRead ( IxQMgrQId qId,
                                               UINT32 *   entryPtr
                                             )
```

Fast read of an entry from a queue.

This function is a heavily streamlined version of **ixQMgrQReadWithChecks()**, but performs essentially the same task. It reads an entire entry from a queue, returning it in entry which must be a pointer to a previously allocated array of sufficient size to hold an entry.

**Note:**

– This function is inlined, to reduce unnecessary function call overhead. It does not perform any parameter checks, or update any statistics. Also, it does not check that the queue specified by `qId` has been configured, or is in range. It simply reads an entry from the queue, and checks for underflow.

– `IX_QMGR_Q_UNDERFLOW` is only returned for queues 0–31 as queues 32–63 do not have an underflow status maintained.

**Parameters:**

*IxQMgrQId(in)* `qId` – the queue identifier.

*UINT32(out)* `*entry` – pointer to the entry word(s).

**Returns:**

◊ `IX_SUCCESS`, entry was successfully read.

◊ `IX_QMGR_Q_UNDERFLOW`, attempt to read from an empty queue

Definition at line **1068** of file **IxQMgr.h**.

References **`IX_QMGR_MIN_QUEUEPP_QID`**, **`IX_QMGR_Q_ENTRY_SIZE1`**, **`IX_QMGR_Q_UNDERFLOW`**, **`IX_SUCCESS`**, **`ixQMgrQReadMWordsMinus1()`**, **`IxQMgrQInlinedReadWriteInfo::qAccRegAddr`**, **`IxQMgrQInlinedReadWriteInfo::qConfigRegAddr`**, **`IxQMgrQInlinedReadWriteInfo::qEntrySizeInWords`**, **`IxQMgrQInlinedReadWriteInfo::qReadCount`**, **`IxQMgrQInlinedReadWriteInfo::qSizeInEntries`**, **`IxQMgrQInlinedReadWriteInfo::qUflowStatBitMask`**, and **`IxQMgrQInlinedReadWriteInfo::qUOStatRegAddr`**.

```
IX_STATUS ixQMgrQReadMWordsMinus1 ( IxQMgrQId qId,
                                     UINT32 * entry
                                   )
```

This function reads the remaining of the `q` entry for queues configured with many words. (the first word of the entry is already read in the inlined function and the entry pointer already incremented).

**Parameters:**

*IxQMgrQId(in)* `qId` – the queue identifier.

*UINT32(out)* `*entry` – pointer to the entry word(s).

**Returns:**

◊ `IX_SUCCESS`, entry was successfully read.

◊ `IX_QMGR_Q_UNDERFLOW`, attempt to read from an empty queue

Referenced by **`ixQMgrQRead()`**.

```

ixQMgrQReadWithChecks ( IxQMgrQId qId,
                        UINT32 *   entry
                        )

```

Read an entry from a queue.

This function reads an entire entry from a queue returning it in entry. The queue configuration word is read to determine what entry size this queue is configured for and then the number of words specified by the entry size is read. entry must be a pointer to a previously allocated array of sufficient size to hold an entry.

**Note:**

– IX\_QMGR\_Q\_UNDERFLOW is only returned for queues 0–31 as queues 32–63 do not have an underflow status maintained.

**Parameters:**

*IxQMgrQId*(in) qId – the queue identifier.

*UINT32*(out) \*entry – pointer to the entry word(s).

**Returns:**

◇ IX\_SUCCESS, entry was successfully read.

◇ IX\_QMGR\_PARAMETER\_ERROR, invalid parameter(s).

◇ IX\_QMGR\_Q\_NOT\_CONFIGURED, queue not configured for this QId

◇ IX\_QMGR\_Q\_UNDERFLOW, attempt to read from an empty queue

```

ixQMgrQShow ( IxQMgrQId qId )

```

Display a queue configuration and statistics for a queue.

This function shows queue configuration and statistics for a queue.

**Parameters:**

*IxQMgrQId* (in)qId – the queue identifier.

**Returns:**

◇ IX\_SUCCESS, success

◇ IX\_QMGR\_Q\_NOT\_CONFIGURED, queue not configured for this QId

```

ixQMgrQSizeInEntriesGet ( IxQMgrQId qId,
                        unsigned *   qSizeInEntries
                        )

```

Return the size of a queue in entries.

This function returns the the size of the queue in entriese.

***Parameters:***

$IxQMgrOID(in)$       qId – the queue identifier

*IxQMgrQSize(out)* \*qSizeInEntries – queue size in entries

**Returns:**

◇ IX\_SUCCESS, successfully retrieved the number of full entries

◊ IX\_QMGR\_Q\_NOT\_CONFIGURED, queue not configured for this QId

◇ IX\_QMGR\_PARAMETER\_ERROR, invalid parameter(s).

```
IX_QMGR_INLINE PUBLIC IX_STATUS ixQMgrQStatusGet ( IxQMgrQId      qId,
                                                    IxQMgrQStatus * qStatus
                                                    )
```

## Fast get of a queue's status.

This function is a streamlined version of `ixQMgrQStatusGetWithChecks()`, but performs essentially the same task. It reads the specified queue's status. A queue's status is defined by its status flags. For queues 0–31 these flags are E, NE, NF, F. For queues 32–63 these flags are NE and F.

**Note:**

- This function is inlined, to reduce unnecessary function call overhead. It does not perform any parameter checks, or update any statistics. Also, it does not check that the queue specified by `qId` has been configured. It simply reads the specified queue's status.

**Parameters:**

$IxOMgrOID(in)$  qId – the queue identifier.

*IxQMgrQStatus(out)* \*qStatus – the status of the specified queue.

**Returns:**

◇ void.

Definition at line 1756 of file IxQMgr.h.

References IX\_QMGR\_MIN\_QUEUPP\_QID, and IX\_SUCCESS.

```
ixQMgrQStatusGetWithChecks ( IxQMgrQId      qId,
                             IxQMgrQStatus * qStatus
                             )
```

Get a queues status.

This function reads the specified queues status. A queues status is defined by its status flags. For queues 0–31 these flags are E,NE,NF,F. For queues 32–63 these flags are NE and F.

**Parameters:**

*IxQMgrQId(in)*      qId – the queue identifier.  
*IxQMgrQStatus(out)* \*qStatus – the status of the specified queue.

**Returns:**

- ◇ IX\_SUCCESS, queue status was successfully read.
- ◇ IX\_QMGR\_Q\_NOT\_CONFIGURED, the specified qId has not been configured
- ◇ IX\_QMGR\_PARAMETER\_ERROR, invalid paramter.

```
IX_QMGR_INLINE PUBLIC IX_STATUS ixQMgrQWrite ( IxQMgrQId qId,
                                              UINT32 *   entry
                                              )
```

Fast write of an entry to a queue.

This function is a heavily streamlined version of **ixQMgrQWriteWithChecks()**, but performs essentially the same task. It will write the entry size number of words pointed to by entry to the queue specified by qId.

**Note:**

- This function is inlined, to reduce unnecessary function call overhead. It does not perform any parameter checks, or update any statistics. Also, it does not check that the queue specified by qId has been configured. It simply writes an entry to the queue, and checks for overflow.
- IX\_QMGR\_Q\_OVERFLOW is only returned for queues 0–31 as queues 32–63 do not have an overflow status maintained.

**Parameters:**

*IxQMgrQId(in)*      qId – the queue identifier.  
*UINT32(in)*      \*entry – pointer to the entry word(s).

**Returns:**

- ◇ IX\_SUCCESS, entry was successfully read.
- ◇ IX\_QMGR\_Q\_OVERFLOW, attempt to write to a full queue

Definition at line **1414** of file **IxQMgr.h**.

References **IX\_QMGR\_MIN\_QUEUEPP\_QID**, **IX\_QMGR\_Q\_ENTRY\_SIZE1**, **IX\_QMGR\_Q\_OVERFLOW**, **IX\_SUCCESS**, **IxQMgrQInlinedReadWriteInfo::qAccRegAddr**, **IxQMgrQInlinedReadWriteInfo::qConfigRegAddr**, **IxQMgrQInlinedReadWriteInfo::qEntrySizeInWords**, **IxQMgrQInlinedReadWriteInfo::qOfFlowStatBitMask**, **IxQMgrQInlinedReadWriteInfo::qSizeInEntries**, **IxQMgrQInlinedReadWriteInfo::qUOStatRegAddr**, and

## **IxQMgrQInlinedReadWriteInfo::qWriteCount.**

```
ixQMgrQWriteWithChecks ( IxQMgrQId qId,  
                          UINT32 *   entry  
                          )
```

Write an entry to an AQM queue.

This function will write the entry size number of words pointed to by entry to the queue specified by qId. The queue configuration word is read to determine the entry size of queue and the corresponding number of words is then written to the queue.

### **Note:**

- IX\_QMGR\_Q\_OVERFLOW is only returned for queues 0–31 as queues 32–63 do not have an overflow status maintained.

### **Parameters:**

*IxQMgrQId(in)* qId – the queue identifier.  
*UINT32(in)* \*entry – the word(s) to write.

### **Returns:**

- ◇ IX\_SUCCESS, value was successfully written.
- ◇ IX\_QMGR\_PARAMETER\_ERROR, invalid paramter(s).
- ◇ IX\_QMGR\_Q\_NOT\_CONFIGURED, queue not configured for this QId
- ◇ IX\_QMGR\_Q\_OVERFLOW, attempt to write to a full queue

```
ixQMgrShow ( void )
```

Describe queue configuration and statistics for active queues.

This function shows active queues, their configurations and statistics.

### **Returns:**

- ◇ void

```
ixQMgrStickyInterruptRegEnable ( void )
```

Enable AQM's sticky interrupt register behaviour only available on B0 Silicon.

When AQM's sticky interrupt register is enabled, interrupt register bit will only be cleared when a '1' is written to interrupt register bit and the interrupting condition is satisfied, i.e.queue condition does not exist.

### **Note:**



This function must be called before any queue is enabled. Calling this function after queue is enabled will cause undefined results.

**Returns:**

none

```
ixQMgrUnload ( void )
```

Uninitialise the QMgr.

This function will perform the tasks required to unload the QMgr component cleanly. This includes unmapping kernel memory. This should be called before a soft reboot or unloading of a kernel module.

**Precondition:**

It should only be called if **ixQMgrInit** has already been called.

**Postcondition:**

No QMgr functions should be called until ixQMgrInit is called again.

**Returns:**

◇ IX\_SUCCESS, the IxQMgr successfully uninitialised

◇ IX\_FAIL, failed to uninitialize the Qmgr

```
ixQMgrWatermarkSet ( IxQMgrQId      qId,  
                     IxQMgrWMLevel ne,  
                     IxQMgrWMLevel nf  
                     )
```

Set the Nearly Empty and Nearly Full Watermarks for a queue.

This function is called by a client to set the watermarks NE and NF for the queue specified by qId. The queue must be empty at the time this function is called, it is the client's responsibility to ensure that the queue is empty. This function will read the status of the queue before the watermarks are set and again after the watermarks are set. If the status register has changed, due to a queue access by an NPE for example, a warning is returned. Queues 32–63 only support the NE flag, therefore the value of nf will be ignored for these queues.

**Parameters:**

*IxQMgrQId(in)* qId – the QId of the queue.

*IxQMgrWMLevel(in)* ne – the NE(Nearly Empty) watermark for this queue. Valid values are 0,1,2,4,8,16,32 and 64 entries.

*IxQMgrWMLevel(in)* nf – the NF(Nearly Full) watermark for this queue. Valid values are 0,1,2,4,8,16,32 and 64 entries.

**Returns:**

◇ IX\_SUCCESS, watermarks have been set for the queue

- ◇ IX\_QMGR\_Q\_NOT\_CONFIGURED, queue not configured for this QId
- ◇ IX\_QMGR\_INVALID\_Q\_WM, invalid watermark
- ◇ IX\_QMGR\_WARNING, the status register may not be consistent

# IXP425 Timer Control (IxTimerCtrl) API

The public API for the IXP425 Timer Control Component.

## Defines

```
#define IX_TIMERCTRL_NO_FREE_TIMERS  
    Timer schedule return code.
```

```
#define IX_TIMERCTRL_PARAM_ERROR  
    Timer schedule return code.
```

## Typedefs

```
typedef  
void(* IxTimerCtrlTimerCallback )(void *userParam)  
    A typedef for a pointer to a timer callback function.
```

## Enumerations

```
enum IxTimerCtrlPurpose {  
    IxTimerCtrlAdslPurpose,  
    IxTimerCtrlMaxPurpose  
}  
    List used to identify the users of timers.
```

## Functions

**IX\_STATUS** **ixTimerCtrlSchedule** (**IxTimerCtrlTimerCallback** func, void \*userParam, **IxTimerCtrlPurpose** purpose, UINT32 relativeTime, unsigned \*timerId)  
*Schedules a callback function to be called after a period of "time". The callback function should not block or run for more than 100ms. This function.*

**IX\_STATUS** **ixTimerCtrlScheduleRepeating** (**IxTimerCtrlTimerCallback** func, void \*param, **IxTimerCtrlPurpose** purpose, UINT32 interval, unsigned \*timerId)  
*Schedules a callback function to be called after a period of "time". The callback function should not block or run for more than 100ms.*

**IX\_STATUS** **ixTimerCtrlCancel** (unsigned id)  
*Cancels a scheduled callback.*

**IX\_STATUS** **ixTimerCtrlInit** (void)  
*Initialise the Timer Control Component.*

void **ixTimerCtrlShow** (void)  
*Display the status of the Timer Control Component.*

---

## Detailed Description

The public API for the IXP425 Timer Control Component.

---

## Define Documentation

```
#define IX_TIMERCTRL_NO_FREE_TIMERS
```

Timer schedule return code.

Indicates that the request to start a timer failed because all available timer resources are used.

Definition at line **84** of file **IxTimerCtrl.h**.

```
#define IX_TIMERCTRL_PARAM_ERROR
```

Timer schedule return code.

Indicates that the request to start a timer failed because the client has supplied invalid parameters.

Definition at line **97** of file **IxTimerCtrl.h**.

---

## Typedef Documentation

```
typedef void(* IxTimerCtrlTimerCallback)(void *userParam)
```

A typedef for a pointer to a timer callback function.

void \* – This parameter is supplied by the client when the timer is started and passed back to the client in the callback.

**Note:**

in general timer callback functions should not block or take longer than 100ms. This constraint is required to ensure that higher priority callbacks are not held up. All callbacks are called from the same thread. This thread is a shared resource. The parameter passed is provided when the timer is scheduled.

Definition at line **117** of file **IxTimerCtrl.h**.

---

# Enumeration Type Documentation

## enum IxTimerCtrlPurpose

List used to identify the users of timers.

**Note:**

The order in this list indicates priority. Components appearing higher in the list will be given priority over components lower in the list. When adding components, please insert at an appropriate position for priority ( i.e values should be less than IxTimerCtrlMaxPurpose ) .

Definition at line **129** of file **IxTimerCtrl.h**.

---

## Function Documentation

### ixTimerCtrlCancel ( unsigned *id* )

Cancels a scheduled callback.

**Parameters:**

*id* – the id of the callback to be cancelled.

**Returns:**

- ◇ IX\_SUCCESS – The timer was successfully stopped.
- ◇ IX\_FAIL – The id parameter did not correspond to any running timer..

**Note:**

This function is re-entrant. The function accesses a list of running timers and may suspend the calling thread if this list is being accessed by another thread.

### ixTimerCtrlInit ( void )

Initialise the Timer Control Component.

**Returns:**

- ◇ IX\_SUCCESS – The timer control component initialized successfully.
- ◇ IX\_FAIL – The timer control component initialization failed, or the component was already initialized.

**Note:**

This must be done before any other API function is called. This function should be called once only and is not re-entrant.

```

ixTimerCtrlSchedule ( IxTimerCtrlTimerCallback func,
                      void * userParam,
                      IxTimerCtrlPurpose purpose,
                      UINT32 relativeTime,
                      unsigned * timerId
                      )

```

Schedules a callback function to be called after a period of "time". The callback function should not block or run for more than 100ms. This function.

**Parameters:**

*func* (in)– the callback function to be called.  
*userParam* (in) – a parameter to send to the callback function, can be NULL.  
*purpose* (in) – the purpose of the callback, internally this component will decide the priority of callbacks with different purpose.  
*relativeTime* (in) – time relative to now in milliseconds after which the callback will be called. The time must be greater than the duration of one OS tick.  
*\*timerId* (out) – An id for the callback scheduled. This id can be used to cancel the callback.

**Returns:**

◇ IX\_SUCCESS – The timer was started successfully.  
 ◇ IX\_TIMERCTRL\_NO\_FREE\_TIMERS – The timer was not started because the maximum number of running timers has been exceeded.  
 ◇ IX\_TIMERCTRL\_PARAM\_ERROR – The timer was not started because the client has supplied a NULL callback func, or the requested timeout is less than one OS tick.

**Note:**

This function is re-entrant. The function accesses a list of running timers and may suspend the calling thread if this list is being accessed by another thread.

```

ixTimerCtrlScheduleRepeating ( IxTimerCtrlTimerCallback func,
                              void * param,
                              IxTimerCtrlPurpose purpose,
                              UINT32 interval,
                              unsigned * timerId
                              )

```

Schedules a callback function to be called after a period of "time". The callback function should not block or run for more than 100ms.

**Parameters:**

*func* (in) – the callback function to be called.  
*userParam* (in) – a parameter to send to the callback function, can be NULL.  
*purpose* (in) – the purpose of the callback, internally this component will decide the priority of callbacks with different purpose.  
*interval* – the interval in milliseconds between calls to func.  
*timerId* (out) – An id for the callback scheduled. This id can be used to cancel the callback.

**Returns:**

- ◇ IX\_SUCCESS – The timer was started successfully.
- ◇ IX\_TIMERCTRL\_NO\_FREE\_TIMERS – The timer was not started because the maximum number of running timers has been exceeded.
- ◇ IX\_TIMERCTRL\_PARAM\_ERROR – The timer was not started because the client has supplied a NULL callback func, or the requested timeout is less than one OS tick.

**Note:**

This function is re-entrant. The function accesses a list of running timers and may suspend the calling thread if this list is being accessed by another thread.

```
ixTimerCtrlShow ( void )
```

Display the status of the Timer Control Component.

**Returns:**

void

**Note:**

Displays a list of running timers. This function is not re-entrant. This function does not suspend the calling thread.

# IXP425 Types (IxTypes)

Basic data types used by the IXP425 project.

## Defines

```
#define OK
#define ERROR
#define NELEMENTS(x)
#define IX_SUCCESS
    Standard return values.

#define IX_FAIL
#define TRUE
    Definition of TRUE.

#define FALSE
    Definition of FALSE.

#define NULL
    definition of NULL

#define PRIVATE
```

## Typedefs

```
typedef void(* IxVoidFnPtr )(void)
typedef void(* IxVoidFnVoidPtr )(void *)
typedef int(* FUNCPTR )(void)
typedef int STATUS
typedef int BOOL
typedef unsigned char UCHAR
typedef unsigned short USHORT
typedef unsigned int UINT
typedef unsigned long ULONG
typedef char INT8
typedef short INT16
typedef int INT32
typedef unsigned char UINT8
typedef unsigned short UINT16
typedef unsigned int UINT32
typedef unsigned long long UINT64
typedef void VOID
typedef volatile UINT32 VUINT32
typedef volatile INT32 VINT32
typedef UINT32 IX_STATUS
```

---



# Detailed Description

Basic data types used by the IXP425 project.

---

## Define Documentation

`#define FALSE`

Definition of FALSE.

Definition at line **133** of file **IxTypes.h**.

`#define IX_SUCCESS`

Standard return values.

Definition at line **115** of file **IxTypes.h**.

Referenced by **ixQMgrQBurstRead()**, **ixQMgrQBurstWrite()**, **ixQMgrQRead()**, **ixQMgrQStatusGet()**, and **ixQMgrQWrite()**.

`#define NULL`

definition of NULL

Definition at line **144** of file **IxTypes.h**.

`#define TRUE`

Definition of TRUE.

Definition at line **122** of file **IxTypes.h**.

# IXP425 UART Access (IxUARTAcc) API

IXP425 UARTAcc Driver Public API.

## Modules

### Defines for Default Values

*Default values which can be used for UART configuration.*

### Defines for IOCTL Commands

*IOCTL Commands (Request codes) which can be used with **ixUARTIoctl**.*

**Defines for IOCTL Arguments** *POSIX style IOCTL arguments which can be used with **ixUARTIoctl**.*

## Data Structures

struct **ixUARTDev** *Device descriptor for the UART.*

struct **ixUARTDev** *Device descriptor for the UART.*

struct **ixUARTStats** *Statistics for the UART.*

struct **ixUARTStats** *Statistics for the UART.*

## Enumerations

```
enum ixUARTMode {  
    INTERRUPT,  
    POLLED,  
    LOOPBACK  
} The mode to set to UART to.
```

## Functions

PUBLIC IX\_STATUS **ixUARTInit** (**ixUARTDev** \*pUART) *Initialise the UART. This puts the chip in a quiescent state.*

PUBLIC IX\_STATUS **ixUARTPollOutput** (**ixUARTDev** \*pUART, int outChar) *Transmit a character in polled mode.*

PUBLIC IX\_STATUS **ixUARTPollInput** (**ixUARTDev** \*pUART, char \*inChar) *Receive a character in polled mode.*

PUBLIC IX\_STATUS **ixUARTIoctl** (**ixUARTDev** \*pUART, int cmd, void \*arg) *Perform I/O control routines on the device.*

---

## Detailed Description

IXP425 UARTAcc Driver Public API.

---

## Enumeration Type Documentation

enum ixUARTMode

The mode to set to UART to.

**Enumeration values:**

*INTERRUPT* Interrupt mode.

*POLLED* Polled mode.

*LOOPBACK* Loopback mode.

Definition at line **317** of file **IxUART.h**.

---

## Function Documentation

IX\_STATUS ixUARTInit ( **ixUARTDev** \* *pUART* )

Initialise the UART. This puts the chip in a quiescent state.

**Parameters:**

*pUART* – pointer to UART structure describing our device.

**Precondition:**

The base address for the UART must contain a valid value. Also the baud rate and hardware options must contain sensible values otherwise the defaults will be used as defined in ixUART.h

**Postcondition:**

UART is initialized and ready to send and receive data.

**Note:**

This function should only be called once per device.

**Return values:**

*IX\_SUCCESS*

– UART device successfully initialised.  
*IX\_FAIL*                      – Critical error, device not initialised.

```

IX_STATUS ixUARTIoctl ( ixUARTDev * pUART,
                        int           cmd,
                        void *        arg
                        )
  
```

Perform I/O control routines on the device.

**Parameters:**

*pUART* – pointer to UART structure describing our device.  
*cmd*    – an ioctl request code.  
*arg*    – optional argument used to set the device mode, baud rate, and hardware options.

**Return values:**

*IX\_SUCCESS* – requested feature was set/read successfully.  
*IX\_FAIL*     – error setting/reading the requested feature.

**See also:**

**IoctlCommandDefines**

**IoctlArgDefines**

```

IX_STATUS ixUARTPollInput ( ixUARTDev * pUART,
                           char *         inChar
                           )
  
```

Receive a character in polled mode.

**Parameters:**

*pUART* – pointer to UART structure describing our device.  
*\*inChar* – character read from the device.

**Precondition:**

UART device must be initialised.

**Return values:**

*IX\_SUCCESS* – character was successfully read.  
*IX\_FAIL*     – input buffer empty (try again).

```
IX_STATUS ixUARTPollOutput ( ixUARTDev * pUART,  
                             int           outChar  
                             )
```

Transmit a character in polled mode.

**Parameters:**

*pUART* – pointer to UART structure describing our device.

*outChar* – character to transmit.

**Precondition:**

UART device must be initialised.

**Return values:**

*IX\_SUCCESS* – character was successfully transmitted.

*IX\_FAIL* – output buffer is full (try again).

# Defines for Default Values

## [IXP425 UART Access (IxUARTAcc) API]

Default values which can be used for UART configuration.

### Defines

**#define IX\_UART\_DEF\_OPTS**

*The default hardware options to set the UART to – no flow control, 8 bit word, 1 stop bit, no parity.*

**#define IX\_UART\_DEF\_XMIT**

*The default UART FIFO size – must be no bigger than 64.*

**#define IX\_UART\_DEF\_BAUD**

*The default UART baud rate – 9600.*

**#define IX\_UART\_MIN\_BAUD**

*The minimum UART baud rate – 9600.*

**#define IX\_UART\_MAX\_BAUD**

*The maximum UART baud rate – 926100.*

**#define IX\_UART\_XTAL**

*The UART clock speed.*

---

## Detailed Description

Default values which can be used for UART configuration.

*See also:*

**ixUARTDev**

---

## Define Documentation

**#define IX\_UART\_DEF\_BAUD**

The default UART baud rate – 9600.

Definition at line **108** of file **IxUART.h**.

**#define IX\_UART\_DEF\_OPTS**

The default hardware options to set the UART to – no flow control, 8 bit word, 1 stop bit, no parity.

Definition at line **90** of file **IxUART.h**.

```
#define IX_UART_DEF_XMIT
```

The default UART FIFO size – must be no bigger than 64.

Definition at line **99** of file **IxUART.h**.

```
#define IX_UART_MAX_BAUD
```

The maximum UART baud rate – 926100.

Definition at line **126** of file **IxUART.h**.

```
#define IX_UART_MIN_BAUD
```

The minimum UART baud rate – 9600.

Definition at line **117** of file **IxUART.h**.

```
#define IX_UART_XTAL
```

The UART clock speed.

Definition at line **135** of file **IxUART.h**.

# Defines for IOCTL Commands

## [IXP425 UART Access (IxUARTAcc) API]

IOCTL Commands (Request codes) which can be used with **ixUARTIoctl**.

### Defines

**#define IX\_BAUD\_SET**  
*Set the baud rate.*

**#define IX\_BAUD\_GET**  
*Get the baud rate.*

**#define IX\_MODE\_SET**  
*Set the UART mode of operation.*

**#define IX\_MODE\_GET**  
*Get the current UART mode of operation.*

**#define IX\_OPTS\_SET**  
*Set the UART device options.*

**#define IX\_OPTS\_GET**  
*Get the UART device options.*

**#define IX\_STATS\_GET**  
*Get the UART statistics.*

---

## Detailed Description

IOCTL Commands (Request codes) which can be used with **ixUARTIoctl**.

---

## Define Documentation

```
#define IX_BAUD_GET
```

Get the baud rate.

Definition at line **165** of file **IxUART.h**.

```
#define IX_BAUD_SET
```



Set the baud rate.

Definition at line **156** of file **IxUART.h**.

```
#define IX_MODE_GET
```

Get the current UART mode of operation.

Definition at line **181** of file **IxUART.h**.

```
#define IX_MODE_SET
```

Set the UART mode of operation.

Definition at line **172** of file **IxUART.h**.

```
#define IX_OPTS_GET
```

Get the UART device options.

Definition at line **199** of file **IxUART.h**.

```
#define IX_OPTS_SET
```

Set the UART device options.

Definition at line **190** of file **IxUART.h**.

```
#define IX_STATS_GET
```

Get the UART statistics.

Definition at line **208** of file **IxUART.h**.

# Defines for IOCTL Arguments

## [IXP425 UART Access (IxUARTAcc) API]

POSIX style IOCTL arguments which can be used with **ixUARTIoctl**.

### Defines

**#define CLOCAL**

*Software flow control.*

**#define CREAD**

*Enable interrupt receiver.*

**#define CSIZE**

*Characters size.*

**#define CS5**

*5 bits*

**#define CS6**

*6 bits*

**#define CS7**

*7 bits*

**#define CS8**

*8 bits*

**#define STOPB**

*Send two stop bits (else one).*

**#define PARENB**

*Parity detection enabled (else disabled).*

**#define PARODD**

*Odd parity (else even).*

---

### Detailed Description

POSIX style IOCTL arguments which can be used with **ixUARTIoctl**.

*See also:*

**ixUARTMode**

---

# Define Documentation

## #define CLOCAL

Software flow control.

Definition at line **230** of file **IxUART.h**.

## #define CREAD

Enable interrupt receiver.

Definition at line **239** of file **IxUART.h**.

## #define CS5

5 bits

Definition at line **257** of file **IxUART.h**.

## #define CS6

6 bits

Definition at line **266** of file **IxUART.h**.

## #define CS7

7 bits

Definition at line **275** of file **IxUART.h**.

## #define CS8

8 bits

Definition at line **284** of file **IxUART.h**.

## #define CSIZE

Characters size.

Definition at line **248** of file **IxUART.h**.

```
#define PARENB
```

Parity detection enabled (else disabled).

Definition at line **302** of file **IxUART.h**.

```
#define PARODD
```

Odd parity (else even).

Definition at line **311** of file **IxUART.h**.

```
#define STOPB
```

Send two stop bits (else one).

Definition at line **293** of file **IxUART.h**.

# IXP400 Version ID (IxVersionId)

Version Identifiers.

## Defines

```
#define IX_VERSION_ID  
    Version Identifier String.  
  
#define IX_VERSION_INTERNAL_ID  
    Internal Release Identifier String.  
  
#define IX_VERSION_COMPATIBLE_TORNADO  
    Compatible Tornado Version Identifier.  
  
#define IX_VERSION_COMPATIBLE_LINUX  
    Compatible Linux Version Identifier.
```

---

## Detailed Description

Version Identifiers.

---

## Define Documentation

```
#define IX_VERSION_COMPATIBLE_LINUX
```

Compatible Linux Version Identifier.

Definition at line **84** of file **IxVersionId.h**.

```
#define IX_VERSION_COMPATIBLE_TORNADO
```

Compatible Tornado Version Identifier.

Definition at line **79** of file **IxVersionId.h**.

```
#define IX_VERSION_ID
```

Version Identifier String.

This string will be updated with each customer release of the IXP400 Software.

Definition at line **66** of file **IxVersionId.h**.

```
#define IX_VERSION_INTERNAL_ID
```

Internal Release Identifier String.

This string will be updated with each internal release (SQA drop) of the IXP400 Software.

Definition at line **74** of file **IxVersionId.h**.

# IXP425 USB Driver Public API

IXP425 USB Driver Public API.

## Data Structures

struct **USBDevice**  
*USBDevice.*

struct **USBSetupPacket**  
*Standard USB Setup packet components, see the USB Specification 1.1.*

## Defines

#define **IX\_USB\_MBLK**  
*Memory buffer.*

#define **IX\_USB\_MBLK\_DATA**(buf)  
*Return pointer to the data in the mbuf.*

#define **IX\_USB\_MBLK\_LEN**(buf)  
*Return pointer to the data length.*

#define **IX\_USB\_MBLK\_FREE**(buf)  
*Returns a buffer to the buffer pool.*

#define **IX\_USB\_MBLK\_PKT\_LEN**(buf)  
*Return pointer to the total length of all the data in the mbuf chain for this packet.*

#define **IX\_USB\_HAS\_GET\_ERROR\_STRING**  
*define to enable ixUSBErrorStringGet()*

#define **IX\_USB\_HAS\_ENDPOINT\_INFO\_SHOW**  
*define to enable ixUSBEndpointInfoShow()*

#define **IX\_USB\_HAS\_STATISTICS\_SHOW**  
*define to enable ixUSBStatisticsShow()*

#define **IX\_USB\_STATS\_SHOW\_PER\_ENDPOINT\_INFO**  
*define to enable per-endpoint information in ixUSBStatisticsShow()*

#define **IX\_USB\_HAS\_VERBOSE\_WARN\_TRACE\_MACRO**  
*define to enable verbose warning tracing*

#define **IX\_USB\_HAS\_CRITICAL\_DATA\_LOCKS**  
*define to enable critical data sections locking*

```

#define IX_USB_HAS_ASSERT_MACRO
    define to enable assertion macro

#define IX_USB_HAS_CT_ASSERT_MACRO
    define to enable compile-time assertion macro

#define IX_USB_HAS_INT_BIND_MACRO
    define to enable interrupt handler binding for VxWorks

#define logMsg
#define UDC_REGISTERS_BASE
    Base I/O address.

#define UDC_IRQ
    IRQ.

#define NUM_ENDPOINTS
    Number of endpoints.

#define SETUP_PACKET_SIZE
    SETUP packet size.

#define CONTROL_FIFO_SIZE
    CONTROL endpoint FIFO depth.

#define CONTROL_PACKET_SIZE
    CONTROL endpoint packet size.

#define INTERRUPT_FIFO_SIZE
    INTERRUPT endpoint FIFO depth.

#define INTERRUPT_PACKET_SIZE
    INTERRUPT endpoint packet size.

#define BULK_FIFO_SIZE
    BULK endpoint FIFO depth.

#define BULK_PACKET_SIZE
    BULK endpoint packet size.

#define ISOCHRONOUS_FIFO_SIZE
    ISOCHRONOUS endpoint FIFO depth.

#define ISOCHRONOUS_PACKET_SIZE
    ISOCHRONOUS endpoint packet size.

#define MAX_TRANSFER_SIZE
    Maximum data size for one transaction in bytes (bulk or control).

#define MAX_QUEUE_SIZE

```



*Maximum outgoing queue size per endpoint, in elements Uses MAX\_QUEUE\_SIZE \* (sizeof(void \*)) bytes.*

**#define MEM\_POOL\_SIZE**

*Memory pool for data transactions.*

**#define TRANSACTION\_TIMEOUT\_RX**

*Maximum acceptable delay in transactions (timestamp ticks), Rx, 0 disables.*

**#define TRANSACTION\_TIMEOUT\_TX**

*Maximum acceptable delay in transactions (timestamp ticks), Tx, 0 disables.*

**#define IX\_USB\_ERROR\_BASE**

*USB error base.*

**#define IX\_USB\_ERROR**

*error due to unknown reasons*

**#define IX\_USB\_INVALID\_DEVICE**

*invalid **USBDevice** structure passed as parameter or no device present*

**#define IX\_USB\_NO\_PERMISSION**

*no permission for attempted operation*

**#define IX\_USB\_REDUNDANT**

*redundant operation*

**#define IX\_USB\_SEND\_QUEUE\_FULL**

*send queue full*

**#define IX\_USB\_NO\_ENDPOINT**

*invalid endpoint*

**#define IX\_USB\_NO\_IN\_CAPABILITY**

*no IN capability on endpoint*

**#define IX\_USB\_NO\_OUT\_CAPABILITY**

*no OUT capability on endpoint*

**#define IX\_USB\_NO\_TRANSFER\_CAPABILITY**

*transfer type incompatible with endpoint*

**#define IX\_USB\_ENDPOINT\_STALLED**

*endpoint stalled*

**#define IX\_USB\_INVALID\_PARMS**

*invalid parameter(s)*

**#define IX\_USB\_DEVICE\_DISABLED**

*device is disabled*

```

#define IX_USB_NO_STALL_CAPABILITY
    no STALL capability

#define EP_DIRECTION(x)
    Macro used to extract the endpoint direction from an EPDescriptorTable[] entry.

#define EP_TYPE(x)
    Macro used to extract the endpoint type from an EPDescriptorTable[] entry.

#define MIN(a, b)
    Compares two values and returns the minimum.

#define MAX(a, b)
    Compares two values and returns the maximum.

#define QUEUE_WRAP(tail)
    Adjusts the tail of a queue implemented in a circular buffer by wrapping at the buffer boundary.

#define SWAP_USB_WORD(wPtr)
    USB byte swapping routine for a little endian platform.

#define REG_GET(reg_ptr)
    read generic register access via register pointers

#define REG_SET(reg_ptr, val)
    write generic register access via register pointers

#define DREG_GET(reg_ptr)
    generic data register read access via register pointers

#define DREG_SET(reg_ptr, val)
    generic data register write access via register pointers

#define CONTEXT(device)
    get context from device pointer

#define REGISTERS(device)
    get registers from device pointer

#define EP0CONTROL(device)
    get endpoint 0 control data from device pointer

#define EVENTS(device)
    get event processor from device pointer

#define COUNTERS(device)
    get device counters

#define OPERATION(device)

```

*get device operation*

```
#define EPSTATUS(device, endpointNumber)
    get endpoint status from device pointer and endpoint number

#define EPQUEUE(device, endpointNumber)
    get endpoint queue from device pointer and endpoint number

#define EPCOUNTERS(device, endpointNumber)
    get endpoint counters from device pointer and endpoint number

#define RETURN_OK(device)
    set IX_SUCCESS on device and return IX_SUCCESS

#define RETURN_ERROR(device)
    set IX_USB_ERROR on device and return IX_FAIL

#define RETURN_INVALID_PARMS(device)
    set IX_USB_INVALID_PARAMS on device and return IX_FAIL

#define RETURN_REDUNDANT(device)
    set IX_USB_REDUNDANT on device and return IX_FAIL

#define RETURN_INVALID_DEVICE(device)
    set IX_USB_INVALID_PARAMS on device and return IX_FAIL

#define RETURN_NO_ENDPOINT(device)
    set IX_USB_INVALID_PARAMS on device and return IX_FAIL

#define RETURN_ENDPOINT_STALLED(device)
    set IX_USB_ENDPOINT_STALLED on device and return IX_FAIL

#define RETURN_SEND_QUEUE_FULL(device)
    set IX_USB_SEND_QUEUE_FULL on device and return IX_FAIL

#define RETURN_NO_IN_CAPABILITY(device)
    set IX_USB_NO_IN_CAPABILITY on device and return IX_FAIL

#define RETURN_NO_STALL_CAPABILITY(device)
    set IX_USB_NO_STALL_CAPABILITY on device and return IX_FAIL

#define RETURN_NO_PERMISSION(device)
    set IX_USB_NO_PERMISSION on device and return IX_FAIL

#define CHECK_DEVICE(device)
    sanity checks for device existence

#define CHECK_DEVICE_ENABLED(device)
    sanity checks for device enable status

#define CHECK_ENDPOINT(device, endpointNumber)
```

*sanity check for endpoint existence*

```
#define CHECK_ENDPOINT_STALL(device, endpointNumber)  
    sanity check for endpoint stall
```

```
#define CHECK_EVENT_MASK(device, eventMask)  
    sanity check for event masks
```

```
#define CHECK_ENDPOINT_QUEUE(epData)  
    sanity check for endpoint queue size
```

```
#define CHECK_ENDPOINT_IN_CAPABILITY(epData, device)  
    sanity check for endpoint IN capability
```

```
#define IX_USB_TRACE  
    no trace macro
```

```
#define IX_USB_LOCK  
    dummy critical data section lock
```

```
#define IX_USB_UNLOCK(state)  
    dummy critical data section unlock
```

```
#define IX_USB_IRQ_LOCK  
    dummy irq lock
```

```
#define IX_USB_IRQ_UNLOCK(state)  
    dummy irq unlock
```

```
#define USB_CONTEXT_SIZE  
    USB context size.
```

## Typedefs

```
typedef UINT16 USBEventSet  
    typedef void(* USBEventCallback )(USBDevice *device, USBEventSet events)  
    typedef void(* USBSetupCallback )(USBDevice *device, const char *packet)  
    typedef void(* USBReceiveCallback )(USBDevice *device, UINT16 sourceEndpoint, IX_USB_MBLK  
        *receiveBuffer)
```

## Enumerations

```
enum USBEndpointDirection {  
    USB_NO_DATA,  
    USB_IN,  
    USB_OUT,  
    USB_IN_OUT  
}
```

*USB endpoint direction.*

```
enum USBEndpointType {  
    USB_CONTROL,  
    USB_BULK,  
    USB_INTERRUPT,  
    USB_ISOCHRONOUS  
}
```

*Note: the values are set for compatibility with USBEndpointDirection.*

```
enum USBEventMap {  
    USB_NO_EVENT,  
    USB_RESET,  
    USB_SUSPEND,  
    USB_RESUME,  
    USB_SOF,  
    USB_DEVICE_EVENTS,  
    USB_BUS_EVENTS,  
    USB_ALL_EVENTS  
}
```

*USB Event Map.*

```
enum USBDeviceFlags {  
    ENABLE_RX_SEQ,  
    ENABLE_TX_SEQ  
}
```

*USB Device Flags.*

```
enum USBEndpointNumber {  
    ENDPOINT_0,  
    ENDPOINT_1,  
    ENDPOINT_2,  
    ENDPOINT_3,  
    ENDPOINT_4,  
    ENDPOINT_5,  
    ENDPOINT_6,  
    ENDPOINT_7,  
    ENDPOINT_8,  
    ENDPOINT_9,  
    ENDPOINT_10,  
    ENDPOINT_11,  
    ENDPOINT_12,  
    ENDPOINT_13,  
    ENDPOINT_14,  
    ENDPOINT_15  
}
```

*USB endpoint number.*

```
enum USBStdRequestType {  
    GET_STATUS_REQUEST,  
    CLEAR_FEATURE_REQUEST,
```

```

SET_FEATURE_REQUEST,
SET_ADDRESS_REQUEST,
GET_DESCRIPTOR_REQUEST,
SET_DESCRIPTOR_REQUEST,
GET_CONFIGURATION_REQUEST,
SET_CONFIGURATION_REQUEST,
GET_INTERFACE_REQUEST,
SET_INTERFACE_REQUEST,
SYNCH_FRAME_REQUEST
}

```

*Standard USB request types.*

```

enum USBStdDescriptorType {
    USB_DEVICE_DESCRIPTOR,
    USB_CONFIGURATION_DESCRIPTOR,
    USB_STRING_DESCRIPTOR,
    USB_INTERFACE_DESCRIPTOR,
    USB_ENDPOINT_DESCRIPTOR
}

```

*Standard USB descriptor types.*

```

enum USBStdFeatureSelector {
    ENDPOINT_STALL,
    DEVICE_REMOTE_WAKEUP
}

```

*Standard USB SET/CLEAR\_FEATURE feature selector.*

```

enum USBStdLanguageId { USB_ENGLISH_LANGUAGE }

```

*Standard language IDs used by USB.*

```

enum USBStdEndpointType {
    USB_CONTROL_ENDPOINT,
    USB_ISOCHRONOUS_ENDPOINT,
    USB_BULK_ENDPOINT,
    USB_INTERRUPT_ENDPOINT
}

```

*Standard USB endpoint types.*

```

enum USBStdEndpointDirection {
    USB_ENDPOINT_OUT,
    USB_ENDPOINT_IN
}

```

*Standard USB directions.*

## Functions

```

PUBLIC
IX_STATUS ixUSBDriverInit (USBDevice *device)

```

*Initialize driver and USB Device Controller.*

**PUBLIC**  
**IX\_STATUS ixUSBDeviceEnable (USBDevice \*device, BOOL enableDevice)**  
*Enable or disable the device.*

**PUBLIC**  
**IX\_STATUS ixUSBEndpointStall (USBDevice \*device, UINT16 endpointNumber, BOOL stallFlag)**  
*Enable or disable endpoint stall (or halt feature).*

**PUBLIC**  
**IX\_STATUS ixUSBEndpointClear (USBDevice \*device, UINT16 endpointNumber)**  
*Free all Rx/Tx buffers associated with an endpoint.*

**PUBLIC**  
**IX\_STATUS ixUSBSignalResume (USBDevice \*device)**  
*Trigger signal resuming on the bus.*

**PUBLIC**  
**IX\_STATUS ixUSBFrameCounterGet (USBDevice \*device, UINT16 \*counter)**  
*Retrieve the 11-bit frame counter.*

**PUBLIC ixUSBReceiveCallbackRegister (USBDevice \*device, USBReceiveCallback**  
**IX\_STATUS callbackFunction)**  
*Register a data receive callback.*

**PUBLIC ixUSBSetupCallbackRegister (USBDevice \*device, USBSetupCallback**  
**IX\_STATUS callbackFunction)**  
*Register a setup receive callback.*

**PUBLIC ixUSBBufferSubmit (USBDevice \*device, UINT16 destinationEndpoint,**  
**IX\_STATUS IX\_USB\_MBLK \*sendBuffer)**  
*Submit a buffer for transmit.*

**PUBLIC ixUSBBufferCancel (USBDevice \*device, UINT16 destinationEndpoint,**  
**IX\_STATUS IX\_USB\_MBLK \*sendBuffer)**  
*Cancel a buffer previously submitted for transmitting.*

**PUBLIC ixUSBEventCallbackRegister (USBDevice \*device, USBEventCallback eventCallback,**  
**IX\_STATUS USBEventMap eventMap)**  
*Register an event callback.*

**PUBLIC ixUSBIsEndpointStalled (USBDevice \*device, UINT16 endpointNumber, BOOL**  
**IX\_STATUS \*stallState)**  
*Retrieve an endpoint's stall status.*

**PUBLIC**  
**IX\_STATUS ixUSBStatisticsShow (USBDevice \*device)**  
*Display device state and statistics.*

**ixUSBErrorStringGet (UINT32 errorCode)**

```

PUBLIC const
    char *
        Convert an error code into a human-readable string error message.

PUBLIC
    IX_STATUS ixUSBEndpointInfoShow (USBDevice *device)
        Display endpoint information table.

int logMsg (char *fmt,...)

```

---

## Detailed Description

IXP425 USB Driver Public API.

---

## Define Documentation

```
#define BULK_FIFO_SIZE
```

BULK endpoint FIFO depth.

Definition at line **98** of file **usbdeviceparam.h**.

```
#define BULK_PACKET_SIZE
```

BULK endpoint packet size.

Definition at line **101** of file **usbdeviceparam.h**.

```
#define CHECK_DEVICE ( device )
```

sanity checks for device existence

Definition at line **277** of file **usbmacros.h**.

```
#define CHECK_DEVICE_ENABLED ( device )
```

sanity checks for device enable status

Definition at line **289** of file **usbmacros.h**.

```
#define CHECK_ENDPOINT ( device,
                        endpointNumber )
```



sanity check for endpoint existence

Definition at line **297** of file **usbmacros.h**.

```
#define CHECK_ENDPOINT_IN_CAPABILITY ( epData,  
                                     device )
```

sanity check for endpoint IN capability

Definition at line **330** of file **usbmacros.h**.

```
#define CHECK_ENDPOINT_QUEUE ( epData )
```

sanity check for endpoint queue size

Definition at line **323** of file **usbmacros.h**.

```
#define CHECK_ENDPOINT_STALL ( device,  
                             endpointNumber )
```

sanity check for endpoint stall

Definition at line **304** of file **usbmacros.h**.

```
#define CHECK_EVENT_MASK ( device,  
                          eventMask )
```

sanity check for event masks

Definition at line **316** of file **usbmacros.h**.

```
#define CONTEXT ( device )
```

get context from device pointer

Definition at line **195** of file **usbmacros.h**.

```
#define CONTROL_FIFO_SIZE
```

CONTROL endpoint FIFO depth.

Definition at line **86** of file **usbdeviceparam.h**.

```
#define CONTROL_PACKET_SIZE
```

CONTROL endpoint packet size.

Definition at line **89** of file **usbdeviceparam.h**.

```
#define COUNTERS ( device )
```

get device counters

Definition at line **207** of file **usbmacros.h**.

```
#define DREG_GET ( reg_ptr )
```

generic data register read access via register pointers

Definition at line **168** of file **usbmacros.h**.

```
#define DREG_SET ( reg_ptr,  
                  val      )
```

generic data register write access via register pointers

Definition at line **170** of file **usbmacros.h**.

```
#define EP0CONTROL ( device )
```

get endpoint 0 control data from device pointer

Definition at line **201** of file **usbmacros.h**.

```
#define EP_DIRECTION ( x )
```

Macro used to extract the endpoint direction from an EPDescriptorTable[] entry.

***Parameters:***

*x* int (in) – the endpoint description entry

***Returns:***

the endpoint direction (USB\_IN, USB\_OUT or USB\_IN\_OUT)

Definition at line **84** of file **usbmacros.h**.

```
#define EP_TYPE ( x )
```

Macro used to extract the endpoint type from an EPDescriptorTable[] entry.

**Parameters:**

*x* int (in) – the endpoint description entry

**Returns:**

the endpoint type (USB\_CONTROL, USB\_BULK, USB\_ISOCHRONOUS, USB\_INTERRUPT)

Definition at line **96** of file **usbmacros.h**.

```
#define EPCOUNTERS ( device,  
                    endpointNumber )
```

get endpoint counters from device pointer and endpoint number

Definition at line **219** of file **usbmacros.h**.

```
#define EPQUEUE ( device,  
                 endpointNumber )
```

get endpoint queue from device pointer and endpoint number

Definition at line **216** of file **usbmacros.h**.

```
#define EPSTATUS ( device,  
                  endpointNumber )
```

get endpoint status from device pointer and endpoint number

Definition at line **213** of file **usbmacros.h**.

```
#define EVENTS ( device )
```

get event processor from device pointer

Definition at line **204** of file **usbmacros.h**.

```
#define INTERRUPT_FIFO_SIZE
```

INTERRUPT endpoint FIFO depth.

Definition at line **92** of file **usbdeviceparam.h**.

```
#define INTERRUPT_PACKET_SIZE
```

INTERRUPT endpoint packet size.

Definition at line **95** of file **usbdeviceparam.h**.

```
#define ISOCHRONOUS_FIFO_SIZE
```

ISOCHRONOUS endpoint FIFO depth.

Definition at line **111** of file **usbdeviceparam.h**.

```
#define ISOCHRONOUS_PACKET_SIZE
```

ISOCHRONOUS endpoint packet size.

Definition at line **114** of file **usbdeviceparam.h**.

```
#define IX_USB_DEVICE_DISABLED
```

device is disabled

Definition at line **105** of file **usberrors.h**.

```
#define IX_USB_ENDPOINT_STALLED
```

endpoint stalled

Definition at line **99** of file **usberrors.h**.

```
#define IX_USB_ERROR
```

error due to unknown reasons

Definition at line **72** of file **usberrors.h**.

```
#define IX_USB_ERROR_BASE
```

USB error base.

Definition at line **68** of file **usberrors.h**.

```
#define IX_USB_HAS_ASSERT_MACRO
```

define to enable assertion macro

Definition at line **134** of file **usbconfig.h**.

```
#define IX_USB_HAS_CRITICAL_DATA_LOCKS
```

define to enable critical data sections locking

Definition at line **130** of file **usbconfig.h**.

```
#define IX_USB_HAS_CT_ASSERT_MACRO
```

define to enable compile-time assertion macro

Definition at line **137** of file **usbconfig.h**.

```
#define IX_USB_HAS_ENDPOINT_INFO_SHOW
```

define to enable **ixUSBEndpointInfoShow()**

Definition at line **72** of file **usbconfig.h**.

```
#define IX_USB_HAS_GET_ERROR_STRING
```

define to enable **ixUSBErrorStringGet()**

Definition at line **69** of file **usbconfig.h**.

```
#define IX_USB_HAS_INT_BIND_MACRO
```

define to enable interrupt handler binding for VxWorks

Definition at line **140** of file **usbconfig.h**.

```
#define IX_USB_HAS_STATISTICS_SHOW
```

define to enable **ixUSBStatisticsShow()**

Definition at line **78** of file **usbconfig.h**.

```
#define IX_USB_HAS_VERBOSE_WARN_TRACE_MACRO
```

define to enable verbose warning tracing

Definition at line **124** of file **usbconfig.h**.

```
#define IX_USB_INVALID_DEVICE
```

invalid **USBDevice** structure passed as parameter or no device present

Definition at line **75** of file **usberrors.h**.

```
#define IX_USB_INVALID_PARMS
```

invalid parameter(s)

Definition at line **102** of file **usberrors.h**.

```
#define IX_USB_IRQ_LOCK
```

dummy irq lock

Definition at line **533** of file **usbmacros.h**.

```
#define IX_USB_IRQ_UNLOCK ( state )
```

dummy irq unlock

Definition at line **536** of file **usbmacros.h**.

```
#define IX_USB_LOCK
```

dummy critical data section lock

Definition at line **527** of file **usbmacros.h**.

```
#define IX_USB_MBLK
```

Memory buffer.

Definition at line **91** of file **usbbasictypes.h**.

```
#define IX_USB_MBLK_DATA ( buf )
```

Return pointer to the data in the mbuf.

Definition at line **94** of file **usbbasictypes.h**.

```
#define IX_USB_MBLK_FREE ( buf )
```

Returns a buffer to the buffer pool.

Definition at line **100** of file **usbbasictypes.h**.

```
#define IX_USB_MBLK_LEN ( buf )
```

Return pointer to the data length.

Definition at line **97** of file **usbbasictypes.h**.

```
#define IX_USB_MBLK_PKT_LEN ( buf )
```

Return pointer to the total length of all the data in the mbuf chain for this packet.

Definition at line **103** of file **usbbasictypes.h**.

```
#define IX_USB_NO_ENDPOINT
```

invalid endpoint

Definition at line **87** of file **usberrors.h**.

```
#define IX_USB_NO_IN_CAPABILITY
```

no IN capability on endpoint

Definition at line **90** of file **usberrors.h**.

```
#define IX_USB_NO_OUT_CAPABILITY
```

no OUT capability on endpoint

Definition at line **93** of file **usberrors.h**.

```
#define IX_USB_NO_PERMISSION
```

no permission for attempted operation

Definition at line **78** of file **usberrors.h**.

```
#define IX_USB_NO_STALL_CAPABILITY
```

no STALL capability

Definition at line **108** of file **usberrors.h**.

```
#define IX_USB_NO_TRANSFER_CAPABILITY
```

transfer type incompatible with endpoint

Definition at line **96** of file **usberrors.h**.

```
#define IX_USB_REDUNDANT
```

redundant operation

Definition at line **81** of file **usberrors.h**.

```
#define IX_USB_SEND_QUEUE_FULL
```

send queue full

Definition at line **84** of file **usberrors.h**.

```
#define IX_USB_STATS_SHOW_PER_ENDPOINT_INFO
```

define to enable per-endpoint information in **ixUSBStatisticsShow()**

Definition at line **100** of file **usbconfig.h**.

```
#define IX_USB_TRACE
```

no trace macro



Definition at line **379** of file **usbmacros.h**.

```
#define IX_USB_UNLOCK ( state )
```

dummy critical data section unlock

Definition at line **530** of file **usbmacros.h**.

```
#define MAX ( a,  
            b )
```

Compares two values and returns the maximum.

**Parameters:**

*a* – first value

*b* – second value

**Returns:**

maximum of the two input values

Definition at line **125** of file **usbmacros.h**.

```
#define MAX_QUEUE_SIZE
```

Maximum outgoing queue size per endpoint, in elements Uses MAX\_QUEUE\_SIZE \* (sizeof(void \*)) bytes.

Definition at line **75** of file **usbdriverparam.h**.

```
#define MAX_TRANSFER_SIZE
```

Maximum data size for one transaction in bytes (bulk or control).

Definition at line **69** of file **usbdriverparam.h**.

```
#define MEM_POOL_SIZE
```

Memory pool for data transactions.

Definition at line **78** of file **usbdriverparam.h**.

```
#define MIN ( a,  
            b )
```

Compares two values and returns the minimum.

**Parameters:**

*a* – first value  
*b* – second value

**Returns:**

minimum of the two input values

Definition at line **110** of file **usbmacros.h**.

```
#define NUM_ENDPOINTS
```

Number of endpoints.

Definition at line **80** of file **usbdeviceparam.h**.

```
#define OPERATION ( device )
```

get device operation

Definition at line **210** of file **usbmacros.h**.

```
#define QUEUE_WRAP ( tail )
```

Ajusts the tail of a queue implemented in a circular buffer by wrapping at the buffer boundary.

**Parameters:**

*tail* int – virtual tail offset

**Returns:**

the real adjusted tail offset

Definition at line **139** of file **usbmacros.h**.

```
#define REG_GET ( reg_ptr )
```

read generic register access via register pointers

Definition at line **163** of file **usbmacros.h**.

```
#define REG_SET ( reg_ptr,  
                val      )
```

write generic register access via register pointers

Definition at line **165** of file **usbmacros.h**.

```
#define REGISTERS ( device )
```

get registers from device pointer

Definition at line **198** of file **usbmacros.h**.

```
#define RETURN_ENDPOINT_STALLED ( device )
```

set IX\_USB\_ENDPOINT\_STALLED on device and return IX\_FAIL

Definition at line **252** of file **usbmacros.h**.

```
#define RETURN_ERROR ( device )
```

set IX\_USB\_ERROR on device and return IX\_FAIL

Definition at line **227** of file **usbmacros.h**.

```
#define RETURN_INVALID_DEVICE ( device )
```

set IX\_USB\_INVALID\_PARAMS on device and return IX\_FAIL

Definition at line **242** of file **usbmacros.h**.

```
#define RETURN_INVALID_PARMs ( device )
```

set IX\_USB\_INVALID\_PARAMS on device and return IX\_FAIL

Definition at line **232** of file **usbmacros.h**.

```
#define RETURN_NO_ENDPOINT ( device )
```

set IX\_USB\_INVALID\_PARAMS on device and return IX\_FAIL

Definition at line **247** of file **usbmacros.h**.

```
#define RETURN_NO_IN_CAPABILITY ( device )
```

set IX\_USB\_NO\_IN\_CAPABILITY on device and return IX\_FAIL

Definition at line **262** of file **usbmacros.h**.

```
#define RETURN_NO_PERMISSION ( device )
```

set IX\_USB\_NO\_PERMISSION on device and return IX\_FAIL

Definition at line **272** of file **usbmacros.h**.

```
#define RETURN_NO_STALL_CAPABILITY ( device )
```

set IX\_USB\_NO\_STALL\_CAPABILITY on device and return IX\_FAIL

Definition at line **267** of file **usbmacros.h**.

```
#define RETURN_OK ( device )
```

set IX\_SUCCESS on device and return IX\_SUCCESS

Definition at line **222** of file **usbmacros.h**.

```
#define RETURN_REDUNDANT ( device )
```

set IX\_USB\_REDUNDANT on device and return IX\_FAIL

Definition at line **237** of file **usbmacros.h**.

```
#define RETURN_SEND_QUEUE_FULL ( device )
```

set IX\_USB\_SEND\_QUEUE\_FULL on device and return IX\_FAIL

Definition at line **257** of file **usbmacros.h**.

```
#define SETUP_PACKET_SIZE
```

SETUP packet size.

Definition at line **83** of file **usbdeviceparam.h**.

```
#define SWAP_USB_WORD ( wPtr )
```

USB byte swapping routine for a little endian platform.

Definition at line **154** of file **usbmacros.h**.

```
#define TRANSACTION_TIMEOUT_RX
```

Maximum acceptable delay in transactions (timestamp ticks), Rx, 0 disables.

Definition at line **81** of file **usbdriverparam.h**.

```
#define TRANSACTION_TIMEOUT_TX
```

Maximum acceptable delay in transactions (timestamp ticks), Tx, 0 disables.

Definition at line **84** of file **usbdriverparam.h**.

```
#define UDC_IRQ
```

IRQ.

Definition at line **77** of file **usbdeviceparam.h**.

```
#define UDC_REGISTERS_BASE
```

Base I/O address.

Definition at line **74** of file **usbdeviceparam.h**.

```
#define USB_CONTEXT_SIZE
```

USB context size.

Definition at line **69** of file **usbtypes.h**.

---

## Enumeration Type Documentation

```
enum USBDeviceFlags
```

USB Device Flags.

Definition at line **108** of file **usbconstants.h**.

## enum USBEndpointDirection

USB endpoint direction.

Definition at line **69** of file **usbconstants.h**.

## enum USBEndpointNumber

USB endpoint number.

Definition at line **117** of file **usbconstants.h**.

## enum USBEndpointType

Note: the values are set for compatibility with USBEndpointDirection.

NB: THESE ARE NOT STANDARD USB ENDPOINT TYPES TO BE USED IN DESCRIPTORS, see **usbstd.h**

Definition at line **82** of file **usbconstants.h**.

## enum USBEventMap

USB Event Map.

***Enumeration values:***

*USB\_SOF* Start Of Frame.

Definition at line **93** of file **usbconstants.h**.

## enum USBStdDescriptorType

Standard USB descriptor types.

Definition at line **90** of file **usbstd.h**.

## enum USBStdEndpointDirection

Standard USB directions.

Definition at line **130** of file **usbstd.h**.

## enum USBStdEndpointType

Standard USB endpoint types.

Definition at line **119** of file **usbstd.h**.

## enum USBStdFeatureSelector

Standard USB SET/CLEAR\_FEATURE feature selector.

Definition at line **102** of file **usbstd.h**.

## enum USBStdLanguageId

Standard language IDs used by USB.

Definition at line **111** of file **usbstd.h**.

## enum USBStdRequestType

Standard USB request types.

Definition at line **72** of file **usbstd.h**.

---

# Function Documentation

```
PUBLIC IX_STATUS ixUSBBufferCancel ( USBDevice *      device,  
                                   UINT16            destinationEndpoint,  
                                   IX_USB_MBLK * sendBuffer  
                                   )
```

Cancel a buffer previously submitted for transmitting.

### **Parameters:**

<i>device</i>	<b>USBDevice</b> * (in) – a structure identifying the device
<i>destinationEndpoint</i>	UINT16 (in) – endpoint originally used for transmitting the data buffer
<i>sendBuffer</i>	IX_USB_MBLK * (in) – submitted data buffer

### **Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBBufferSubmit ( USBDevice * device,
                                   UINT16 destinationEndpoint,
                                   IX_USB_MBLK * sendBuffer
                                   )
```

Submit a buffer for transmit.

**Parameters:**

*device* **USBDevice** \* (in) – a structure identifying the device  
*destinationEndpoint* UINT16 (in) – endpoint to be used for transmitting the data buffer  
*sendBuffer* IX\_USB\_MBLK \* (in) – data buffer

**Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBDeviceEnable ( USBDevice * device,
                                   BOOL enableDevice
                                   )
```

Enable or disable the device.

**Parameters:**

*device* **USBDevice** \* (in) – a structure identifying the device  
*enableDevice* BOOL (in) – **true** to enable the device and **false** to disable it

This function enables or disables the device. A disabled device doesn't generate events and cannot send or receive data.

Disabling the device frees and discards all existent Rx/Tx buffers (received buffers that weren't dispatched yet and buffers waiting to be transmitted)

**Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBDriverInit ( USBDevice * device )
```

Initialize driver and USB Device Controller.

**Parameters:**

*device* **USBDevice** \* (inout) – a structure identifying the device

This function initializes the UDC and all the data structures used to interact with the controller.

It is the responsibility of the caller to create the **USBDevice** structure and fill in the correct *baseIOAddress* and *interruptLevel* fields.

After successful initialization the device will be inactive – use **ixUSBDeviceEnable** to activate the device. Use the *flags* component of the *device* structure to pass in additional flags such as ENABLE\_RX\_SEQ or



ENABLE\_TX\_SEQ. Changing these flags later will have no effect.

The driver will assign a device number which will be placed in the *deviceIndex* field.

The initialized *device* structure must be used for all interactions with the USB controller. The same *device* pointer will be passed in to all the registered client callbacks.

The *deviceIndex* and *deviceContext* should be treated as read-only fields. A check to verify that the USB device is present is performed and a warning is issued if the device is not present.

**Warning:**

This function is not reentrant.

**Returns:**

IX\_SUCCESS if the initialization was successful; a warning is issued if the specified USB device is not present. IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBEndpointClear ( USBDevice * device,
                                     UINT16    endpointNumber
                                   )
```

Free all Rx/Tx buffers associated with an endpoint.

**Parameters:**

*device*                    **USBDevice \*** (in) – a structure identifying the device  
*endpointNumber*   **UINT16** (in) – endpoint number

This function discards and frees all Tx/Rx buffers associated with an endpoint. The corresponding endpoint dropped packet counters will also be incremented.

**Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC void ixUSBEndpointInfoShow ( USBDevice * device )
```

Display endpoint information table.

**Parameters:**

*device*                    **USBDevice \*** (in) – a structure  
                             identifying the device

**Returns:**

none

```
PUBLIC IX_STATUS ixUSBEndpointStall ( USBDevice * device,
                                     UINT16    endpointNumber,
                                     BOOL        stallFlag
                                   )
```

Enable or disable endpoint stall (or *halt* feature).

***Parameters:***

<i>device</i>	<b>USBDevice</b> * (in) – a structure identifying the device
<i>endpointNumber</i>	UINT16 (in) – endpoint number
<i>stallFlag</i>	BOOL (in) – <b>true</b> to set endpoint stall and <b>false</b> to clear it

This function clears or sets the endpoint stall (or *halt*) feature.

Both IN and OUT endpoints can be stalled. A stalled endpoint will not send or receive data. Instead, it will send USB STALL packets in response to IN or OUT tokens.

Unstalling endpoints can be done only by using this function with the exception of endpoint 0 which uninstalls itself automatically upon receiving a new SETUP packet, as required by the USB 1.1 Specification. Isochronous endpoints cannot be stalled and attempting to do so will return an `IX_USB_NO_STALL_CAPABILITY` failure.

**Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

PUBLIC const char \* ixUSBErrorStringGet ( UINT32 errorCode )

Convert an error code into a human-readable string error message.

***Parameters:***

*errorCode* UINT32 (in) – error code as defined in **usberrors.h**

**Returns:**

```
a const char * pointer to the error message
```

```

PUBLIC IX_STATUS ixUSBEventCallbackRegister ( USBDevice *      device,
                                              USBEventCallback eventCallback,
                                              USBEventMap      eventMap
                                              )

```

Register an event callback.

***Parameters:***

<i>device</i>	<b>USBDevice</b> * (in) – a structure identifying the device
<i>eventCallback</i>	USBEventCallback (in) – event callback function
<i>eventMap</i>	USBEventMap (in) – event map

**Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBFrameCounterGet ( USBDevice * device,
                                         UINT16 * counter
                                         )
```

Retrieve the 11-bit frame counter.

***Parameters:***

*device* **USBDevice** \* (in) – a structure identifying the device  
*counter* **UINT16** \* (out) – the 11-bit frame counter

This function returns the hardware USB frame counter.  
Since the counter is 11-bit wide it rolls over after every 2048 frames.

**Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBIsEndpointStalled ( USBDevice * device,
                                           UINT16      endpointNumber,
                                           BOOL *        stallState
                                           )
```

Retrieve an endpoint's stall status.

**Parameters:**

<i>device</i>	<b>USBDevice</b> * (in) – a structure identifying the device
<i>endpointNumber</i>	UINT16 (in) – endpoint number
<i>stallState</i>	BOOL * (out) – stall state; <b>true</b> if the endpoint is stalled ( <i>halted</i> ) or <b>false</b> otherwise <b>true</b>

**Returns:**

IX\_SUCCESS or IX\_FAIL if the device pointer is invalid or the endpoint doesn't exist

```

PUBLIC IX_STATUS ixUSBReceiveCallbackRegister ( USBDevice *      device,
                                                USBReceiveCallback callbackFunction
                                                )

```

Register a data receive callback.

***Parameters:***

<i>device</i>	<b>USBDevice</b> * (in) – a structure identifying the device
<i>callbackFunction</i>	<b>USBReceiveCallback</b> (in) – receive callback function

**Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBSetupCallbackRegister ( USBDevice * device,
                                             USBSetupCallback callbackFunction
                                             )
```

Register a setup receive callback.

**Parameters:**

*device*                    **USBDevice** \* (in) – a structure identifying the device  
*callbackFunction*    USBSetupCallback (in) – setup callback function

**Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBSignalResume ( USBDevice * device )
```

Trigger signal resuming on the bus.

**Parameters:**

*device*    **USBDevice** \* (in) – a structure identifying the device

This function triggers signal resuming on the bus, waking up the USB host. It should be used only if the host has enabled the device to do so using the standard SET\_FEATURE USB request, otherwise the function will return IX\_FAIL and set the *lastError* field to IX\_USB\_NO\_PERMISSION.

**Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

```
PUBLIC IX_STATUS ixUSBStatisticsShow ( USBDevice * device )
```

Display device state and statistics.

**Parameters:**

*device*    **USBDevice** \* (in) – a structure identifying the device

**Returns:**

IX\_SUCCESS if the initialization was successful, IX\_FAIL otherwise, in which case a detailed error code will be set in the *lastError* field, unless the *device* parameter is NULL.

# IXP425 Codelets

IXP425 Codelets.

## Modules

### **IXP425 ATM Codelet (IxAtmCodelet) API**

*This codelet demonstrates an example implementation of a working Atm driver that makes use of the AtmdAcc component, as well as demonstrating how the lower layer IxAtmdAcc component can be used for configuration and control.*

### **IXP425 DMA Access Codelet (IxDmaAccCodelet) API**

*IXP425 DMA Access component API.*

### **IXP425 Ethernet Aal5 (IxEthAal5App) API** *IXP425 Ethernet Aal5 Codelet component API.*

### **IXP425 Ethernet Access Codelet (IxEthAccCodelet) API** *IXP425 Ethernet Access Codelet API.*

### **IXP425 HSS Access Codelet (IxHssAccCodelet) API** *IXP425 HSS Access Codelet API. The interface for the HSS Access Codelet.*

### **IXP425 PerfProf Access Codelet (IxPerfProfAccCodelet) API** *IXP425 codelet PerfProf Access component API*

### **IXP425 Timers (IxTimersCodelet) API** *IXP425 Timer component API.*

### **IXP425 USB RNDIS Codelet (IxUSBRNDIS) API** *IXP425 codelet USB RNDIS API.*

---

## Detailed Description

IXP425 Codelets.

# IXP425 ATM Codelet (IxAtmCodelet) API

## [IXP425 Codelets]

This codelet demonstrates an example implementation of a working Atm driver that makes use of the AtmdAcc component, as well as demonstrating how the lower layer IxAtmdAcc component can be used for configuration and control.

## Defines

```
#define IX_ATM_CODELET_SWLOOPBACK_PORT_RATE
    Port rate for Software Loopback.

#define IX_ATM_CODELET_REMOTELOOPBACK_PORT_RATE
    Port rate for Remote Loopback.

#define IX_ATMCODELET_START_VPI
    The first VPI value.

#define IX_ATMCODELET_START_VCI
    The first VCI value.

#define IX_ATMCODELET_NUM_AAL5_CHANNELS
    32 Channels for AAL5

#define IX_ATMCODELET_NUM_AAL0_48_CHANNELS
    32 Channels for AAL0 48-bytes

#define IX_ATMCODELET_NUM_AAL0_52_CHANNELS
    32 Channels for AAL0 52-bytes
```

## Functions

```
PUBLIC IxAtmCodeletMain (IxAtmCodeletMode modeType, IxAtmCodeletAalType
IX_STATUS aalType)
    This function is used as a single point of execution for ATM codelet.
```

---

## Detailed Description

This codelet demonstrates an example implementation of a working Atm driver that makes use of the AtmdAcc component, as well as demonstrating how the lower layer IxAtmdAcc component can be used for configuration and control.

This codelet also demonstrates an example implementation of OAM F4 Segment, F4 End-To-End (ETE), F5

Segment and F5 ETE loopback that makes use of the AtmdAcc component, as well as demonstrating how the lower layer IxAtmdAcc component can be used for configuration and control.

#### **Disclaimer Note:** For Linux Platform

- When 'insmod' the ATM codelet object, it will begin to send AAL packets and display the transmit and receive statistics every 15 second
- Unable to 'rmmod'. User will not be able to type anything in the command line due to: a) the return carriage indicating that the task is sending AAL packets, and b) the failure to kill the thread which is used to transmit AAL packets

#### **VxWorks User Guide**

**ixAtmCodeletMain()** function is used as a single point of execution for Atm Codelet, which allows the user to enter selections for different type of modes and AAL type. The function also allows the user to execute OAM ping either in UTOPIA or Software Loopback mode. In all modes, the transmit and receive statistics will be displayed every 15secs.

```
Usage :
    ixAtmCodeletMain (modeType, aalType)
    modeType:
        0 = Utopia Loopback Mode
        1 = Software Loopback Mode
        2 = Remote Loopback Mode
        3 = F4 & F5 cells OAM Ping in UTOPIA Loopback mode
        4 = F4 & F5 cells OAM Ping in Software Loopback mode
    aalType:
        1 = AAL5
        2 = AAL0_48
        3 = AAL0_52
```

#### **Linux User Guide**

The idea of using the **ixAtmCodeletMain()** as a single point of execution for ATM codelet is similar in VxWorks User Guide. It also allows user to execute OAM ping. Similarly, all modes will display the transmit and receive statistics every 15secs. This function will be executed when user issue 'insmod' in command prompt

```
Usage :
    # insmod ixp400_codelets_atm.o modeType= aalType=
    Where x:
        0 = Utopia Loopback Mode
        1 = Software Loopback Mode
        2 = Remote Loopback Mode
        3 = F4 & F5 cells OAM Ping in UTOPIA Loopback mode
        4 = F4 & F5 cells OAM Ping in Software Loopback mode
    Where y:
        1 = AAL5
        2 = AAL0_48
        3 = AAL0_52
```

Note for VxWorks and Unix Usage:

- IX\_ATM\_CODELET\_SWLOOPBACK\_PORT\_RATE and IX\_ATM\_CODELET\_REMOTELOOPBACK\_PORT\_RATE defined in this header file allows the user to change the port rate (in cells/sec) accordingly. The port rate works when using ADSL connection. The default port rate for both loopback is set to 1962cells/sec (~832kbps)
- IX\_ATM\_CODELET\_START\_VPI and IX\_ATM\_CODELET\_START\_VCI defined in this header file can be modified by the user. By default, VPI and VCI are set to 1 and 32 respectively
- IX\_ATM\_CODELET\_NUM\_AAL5\_CHANNELS, IX\_ATM\_CODELET\_NUM\_AAL0\_48\_CHANNELS, and IX\_ATM\_CODELET\_NUM\_AAL0\_52\_CHANNELS can be changed by the user. By default, they are set to 32 channels
- OAM Ping in UTOPIA Loopback mode will perform the following sequence in forever loop: i) Send AAL packets, ii) Display the transmit and receive statistics, and iii) Perform OAM Ping F4 and F5 (ETE and Segment) and display OAM statistics
- OAM Ping in Software Loopback will perform the following sequence in forever loop: i) Display the transmit and receive statistics, and ii) OAM Ping F4 and F5 (ETE and Segment) and display OAM Statistics

## ATM Features

- An interface is provided to setup Aal5 or Aal0 (48 or 52 bytes) Transmit and Recieve VCs on one port. Only UBR VCs can be setup. When a channel is setup using this interface both a Transmit and a Recieve VC is created.
- An interface is provided to remove all registered Aal5/Aal0 VCs.
- Both remote and local loopback of Aal5 or Aal0 is provided by this codelet. Local loopback refers to loopback provided by the UTOPIA interface. In local loopback packets generated by the IXP425 are looped back to the Atm driver by the UTOPIA hardware. Software loopback refers to the software looping all packets received on the wire back out onto the wire. Remote loopback refers to where the far end is expected to perform a software loopback, i.e. any packets sent by the codelet are expected to be looped back by the far end into the codelet.
- Both interrupt and polled mode of operation is provided by this codelet.
- An interface is provided to send Aal5 SDUs specifying the sdu size and the number of Aal5 sdus to send.
- An interface is provided to send Aal0 packets specifying the packet size and the number of Aal0 packets to send.
- Both the Transmit port rate and the Recieve port rate can be modified. The Tx port rate is used by IxAtmSch in performing the shaping functions The Rx port rate is not used by any component.
- An interface is provided to allow the querying of the ATM ports and registered VCs.

## IXP425 ATM Components used by this codelet

- IxAtmdAcc. This component is the low level interface by which AAL packets get transmitted to, and received from the UTOPIA bus.

## IXP425 ATM Codelet components used by this codelet



- **IxAtmSch.** This component demonstrates an ATM Traffic Shaper implementation. IxAtmdAcc gets scheduling information from this component.
- **IxAtmm.** This component provides ATM port and VC management facilities. This component also manages the configuration of the UTOPIA co-processor

### **IxAtmCodelet modes of operation**

This codelet can be initialised to operate in one of three configurations.

- **IX\_ATMCODELET\_UTOPIA\_LOOPBACK.** In this mode the UTOPIA interface will loopback all traffic transmitted.

**Buffer management** In this mode a simple buffering mechanism is used; mbufs are allocated from the vxWorks pool as needed for RxFree replenishing/Tx and are returned to the vxWorks pool for TxDone/Rx

**Interrupt/Task based processing** In this mode of operation the IxQMGrDispatcher is hooked to interrupts. This means that TxDone/RxLo will be performed from a task level and IxAtmdAcc Tx processing/RxHi and RxFree will be precessed from an interrupt level.

**Sending Aal5 PDUs** In this mode of operation an interface is available to send Aal5 PDUs of specified size in bytes.

**Sending Aal0 Packets** In this mode of operation an interface is available to send Aal0 packets of specified size in cells.

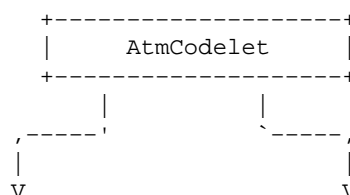
- **IX\_ATMCODELET\_REMOTE\_LOOPBACK.** In this mode packets are sent and are expected to be looped back on the remote end.
- **IX\_ATMCODELET\_SOFTWARE\_LOOPBACK.** In this mode of operation all traffic sent are looped back to receive in the IXP425 device.

**Buffer management** In this mode a more complex buffering mechanism is used; mbufs are allocated from the vxWorks pool and stored in a software queue. These mbufs are fetched from this software queue as needed for RxFree replenishing/Tx and are returned to the software queue for TxDone/Rx.

**Interrupt/Task based processing** In this mode of operation the IxQMGrDispatcher is called form a task every 1 msec. This means that TxDone/RxLo/IxAtmdAcc Tx processing/RxHi and RxFree will be processed from a task level.

**Sending PDUs/Packets** This mode of operation does not provide an interface for sending Aal5 PDUs/Aal0 Packets.

### **IxAtmCodelet sub-components**





**AtmCodelet** This implements the API functions.

**RxTx** This sub-component implements the IX\_ATMCODELET\_UTOPIA\_LOOPBACK and IX\_ATMCODELET\_REMOTE\_LOOPBACK modes of operation.

**SwLoopback** This sub-component implements the IX\_ATMCODELET\_SOFTWARE\_LOOPBACK mode operation.

**BufMan** This sub-component implements the interfaces used internally for getting and returning vxWorks mbufs.

**UTOPIA Recieve and Transmit PHY addresses** In this codelet UTOPIA Phy Addresses are assigned numbers starting at UTOPIA\_PHY\_ADDR and are incremented for each port used. These addresses \_WILL\_ need to be changed depending on the hardware setup.

## OAM Features

- An interface is provided to configure the access layer to enable OAM traffic.
- OAM Loopback responses are sent automatically on receipt of OAM F4 and F5 parent Segment and ETE loopback cells, i.e. externally initiated OAM Loopbacks.
- Interfaces are provided to initiate an OAM F4/F5 Segment or ETE Loopback, see ITU-610. One loopback can be initiated at a time, this is to keep the codelet simple, this is independant of sending responses to parent loopback cells received.
- An interface is provided to query OAM traffic statistics.

## OAM uses the following IXP425 ATM Components

- IxAtmdAcc. This component is the low level interface by which OAM cells get transmitted to, and received from the UTOPIA bus.
- IxAtmm. This component provides ATM port and VC management facilities. This component also contains the configuration of the UTOPIA co-processor.
- AtmUtils. This codelet provides VC setup facilities using IxAtmdAcc.

## IMPORTANT!!!

If validation board is used (instead of using IXDP425 development board), then uncomment VALIDATION\_PLATFORM\_USED flag in component.mk

---

# Define Documentation

```
#define IX_ATM_CODELET_REMOTELOOPBACK_PORT_RATE
```

Port rate for Remote Loopback.

By default the port rate for remote loopback is set at PCR = 1962 cells/sec (832kbps)

Definition at line **348** of file **IxAtmCodelet.h**.

```
#define IX_ATM_CODELET_SWLOOPBACK_PORT_RATE
```

Port rate for Software Loopback.

By default the port rate for software loopback is set at PCR = 1962 cells/sec (832kbps)

Definition at line **335** of file **IxAtmCodelet.h**.

```
#define IX_ATMCODELET_NUM_AAL0_48_CHANNELS
```

32 Channels for AAL0 48-bytes

Definition at line **388** of file **IxAtmCodelet.h**.

```
#define IX_ATMCODELET_NUM_AAL0_52_CHANNELS
```

32 Channels for AAL0 52-bytes

Definition at line **397** of file **IxAtmCodelet.h**.

```
#define IX_ATMCODELET_NUM_AAL5_CHANNELS
```

32 Channels for AAL5

Definition at line **379** of file **IxAtmCodelet.h**.

```
#define IX_ATMCODELET_START_VCI
```

The first VCI value.

By default the VCI is set to 2

Definition at line **370** of file **IxAtmCodelet.h**.

```
#define IX_ATMCODELET_START_VPI
```

The first VPI value.

By default the VPI is set to 1

Definition at line **359** of file **IxAtmCodelet.h**.

---

## Function Documentation

```
ixAtmCodeletMain ( IxAtmCodeletMode    modeType,  
                  IxAtmCodeletAalType aalType  
                  )
```

This function is used as a single point of execution for ATM codelet.

### **Parameters:**

<i>IxAtmCodeletMode</i> <i>modeType</i>	The type of mode use, either Utopia, Software or Remote loopback mode. It also consists of OAM Ping in Utopia or Software loopback mode.
<i>IxAtmCodeletAalType</i> <i>aalType</i>	The type of AAL: AAL5 or AAL0 with 48– or 52–bytes cell

### **Returns:**

IX\_SUCCESS Mode and AAL type successfully setup  
IX\_FAIL Invalid AAL or mode type, or error in setting up the modes

# IXP425 DMA Access Codelet (IxDmaAccCodelet) API

## [IXP425 Codelets]

IXP425 DMA Access component API.

## Defines

#define **IX\_DMA\_CODELET\_TRANSFER\_LENGTH**  
*The length of the transfer size if 128 bytes.*

## Functions

IX\_STATUS **ixDmaAccCodeletMain** (void)

*This function is the entry point to the Dma Access codelet. It will initialise the Dma codelet which in turn initialises the necessary components.*

---

## Detailed Description

IXP425 DMA Access component API.

This file contains a main interface of the Dma Access Codelet that initialises the DmaAcc codelet and execute Dma transfer using **ixDmaAccCodeletTestPerform()** function for various DMA transfer mode, addressing mode and transfer width. The block size used in this codelet are 8,1024,16384,32768,65528 bytes. For each Dma configuration, the performance will be measured and the average rate (in Mbps) will be displayed

### VxWorks User Guide

Usage :  
-> **ixDmaAccCodeletMain()**

Note:

1. Once the function is executed, the codelet will display the results

2. The formulae to calculate the rate is:

$$\text{Rate (in Mbps)} = ( (\text{length} * 8) / (\text{ticks} / 66) )$$

### Linux User Guide

```
Usage :  
# insmod ixp400_codelets_dmaAcc.o
```

Note:

1. Once the function is executed, the codelet will display the results

2. The formulae to calculate the rate is:

$$\text{Rate (in Mbps)} = ( (\text{length} * 8) / (\text{ticks} / 66) )$$

## DmaAcc Codelet Features

The API **ixDmaAccCodeletTestPerform()** allows the user to perform a Dma transfer of block size 0 to 65535 bytes between two locations in the SRAM. The user can specify any combination of the following modes.

DMA Transfer Modes 1. Copy 2. Copy and Clear Source 3. Copy with Bytes Swap 4. Copy with Bytes Reversed

DMA Addressing Modes 1. Incremental Source to Incremental Destination Addressess 2. Fixed Source to Incremental Destination Addressess 3. Incremental Source to Fixed Destination Addressess

DMA Transfer Widths 1. 32-bit Transfer 2. 16-bit Transfer 3. 8-bit Transfer 4. Burst Transfer

NOTE : The user must initialise the system with **ixDmaAccCodeletInit** prior to calling the function **ixDmaAccCodeletTestPerform()**

Performance will execute **PERFORMANCE\_NUM\_LOOP** (i.e. 100 runs) in order to calculate the average rate for each Dma transfer configuration

---

## Define Documentation

```
#define IX_DMA_CODELET_TRANSFER_LENGTH
```

The length of the transfer size if 128 bytes.

It can be changed for Dma transfer. The range is between 1–65535 bytes

Definition at line **150** of file **IxDmaAccCodelet.h**.

---

# Function Documentation

```
void ixDmaAccCodeletMain ( void )
```

This function is the entry point to the Dma Access codelet. It will initialise the Dma codelet which in turn initialises the necessary components.

Once it has successfully initialise the Dma Codelet, this function will continue to perform valid DMA transfer using IxDmaAccCodeletTestPerform()

***Parameters:***

*none*

***Returns:***

*none*

# IXP425 Ethernet Aal5 (IxEthAal5App) API

## [IXP425 Codelets]

IXP425 Ethernet Aal5 Codelet component API.

### Defines

```
#define IX_EAA_NUM_BUFFERS_PER_ETH
    This is the number of buffers, which can be stored in free buffer queue for each ethernet port.

#define IX_EAA_NUM_ATM_PORTS
    Define number of supported atm ports by this application.

#define IX_EAA_NUM_BUFFERS_PER_VC
    This is the number of buffers per atm port.

#define IX_EAA_PORT1_VPI
    Define default VPI and VCI for 8 ports. User can define them in EthAal5User.h file to overwrite definitions below.

#define IX_EAA_PORT1_VCI
#define IX_EAA_PORT2_VPI
#define IX_EAA_PORT2_VCI
#define IX_EAA_PORT3_VPI
#define IX_EAA_PORT3_VCI
#define IX_EAA_PORT4_VPI
#define IX_EAA_PORT4_VCI
#define IX_EAA_PORT5_VPI
#define IX_EAA_PORT5_VCI
#define IX_EAA_PORT6_VPI
#define IX_EAA_PORT6_VCI
#define IX_EAA_PORT7_VPI
#define IX_EAA_PORT7_VCI
#define IX_EAA_PORT8_VPI
#define IX_EAA_PORT8_VCI
#define IX_EAA_MAC1
    Define default for MAC1 address for ixEAAAddMac() function.

#define IX_EAA_MAC2
    Define default for MAC2 address for ixEAAAddMac() function.

#define IX_EAA_MAC3
    Define default for MAC3 address for ixEAAAddMac() function.

#define IX_EAA_MAC4
    Define default for MAC4 address for ixEAAAddMac() function.
```



```
#define IX_EAA_MAC5
    Define default for MAC5 address for ixEAAAddMac() function.

#define IX_EAA_MAC6
    Define default for MAC6 address for ixEAAAddMac() function.

#define IX_EAA_PORT0
    Define default for Port number for ixEAAAddMac() function.
```

## Functions

```
PUBLIC
IX_STATUS ixEthAal5AppCodeletMain (IxEAAModeType modeType)
    This is the main function that executes the EthAal5App codelet.
```

---

## Detailed Description

IXP425 Ethernet Aal5 Codelet component API.

IxEthAal5App application is also called as IXP425 Mini Bridge application which bridges traffic between Ethernet and Utopia ports or Ethernet and ADSL ports. It uses ixEthAcc, ixAtmdAcc, ixAtmm, ixAtmSch and ixQmgr software components.

### VxWorks User Guide

**ixEthAal5AppCodeletMain()** function is used as a single point of execution for EthAal5 Codelet, which allows the user to enter in 2 type of modes: Utopia or ADSL, in order to operate together with ethernet.

```
Usage :
    ixEthAal5AppCodeletMain (modeType)
    modeType:
        1 = Utopia
        2 = ADSL
```

### Linux User Guide

**ixEthAal5AppCodeletMain()** function is used as a single point of execution for EthAal5 Codelet, which allows the user to enter in 2 type of modes: Utopia or ADSL, in order to operate together with ethernet.

```
Usage :
    # insmod ixp400_codelets_ethAal5App.o modeType=
    Where x:
        1 = Utopia
        2 = ADSL
```

Note for VxWorks and Linux Usage: In order to observe the current traffic counters, the ixEAAShow() function is executed every 15seconds. This applies to both Utopia and ADSL mode.

## Features

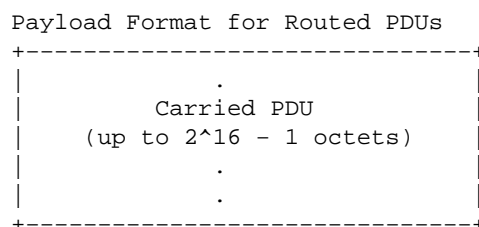
This Application currently supports 2 Ethernet ports and up to 8 Utopia phys, which are initialized by default at the start of application. Ethernet frames are transferred across ATM link (through Utopia interface) using AAL5 protocol and Ethernet frame encapsulation described by RFC 1483. MAC address learning is performed on Ethernet frames, received by Ethernet ports and ATM interface (encapsulated). Application filters packets basing on destination MAC addresses – packets are forwarded to other port only if the port has ever received packet/frame with the same source MAC address. Forwarding is done only between Ethernet and Utopia port. Two simplifications were made to keep code simple:

- ◇ Application doesn't allow packet forwarding between Ethernet ports (nor Utopia ports).
- ◇ flooding (forwarding frames/packets with unknown MAC addresses) is not supported. Two IxEthAal5App will never transfer any packets between each other, because initially MAC data base is empty, so all packets will be filtered out. However there is function ixEAAAddMAC which can be used to add MAC address to the data base and assign it to one of available ports. To enable simplified flooding see comments in ixEAAEthRxCallback.
- ◇ This application can not be executed more than once. It doesn't deinitialize itself. If user wishes to change configuration and run application again the whole system (vxWorks) must be restarted.
- ◇ currently Mac Learning/Filtering database in ixEthAcc component supports only Ethernet ports. For that reason it couldn't be used in this application for learning Mac addresses from encapsulated Ethernet frames received from Utopia. In the near future ixEthAcc component will support all possible ports (including Utopia), but by this time a very simplified approach is used in this application: only one Mac address is stored per VC (and there is one VC per Phy). It means, that only one Mac address is supported simultaneously per Phy. This is done to keep code as simple as possible.
- ◇ This application doesn't initialize any DSL connection. It initializes Utopia interface – if any DSL card is attached to the Richfield board it should be initialized before start of application (e.g. to initialize Adsl card call 'adslUp' function from windshell – adsl module must be separately loaded).
- ◇ 2 protocols from RFC 1483 are recognized: The first packet received from ATM will decide the behaviour of the application (ether bridged or routed)

## From RFC 1483

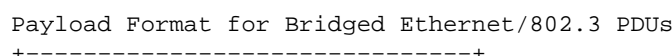
## VC Based Multiplexing of Routed Protocols

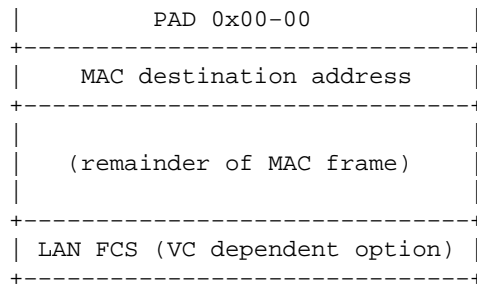
PDUs of routed protocols shall be carried as such in the Payload of the AAL5 CPCS-PDU. The format of the AAL5 CPCS-PDU Payload field thus becomes:



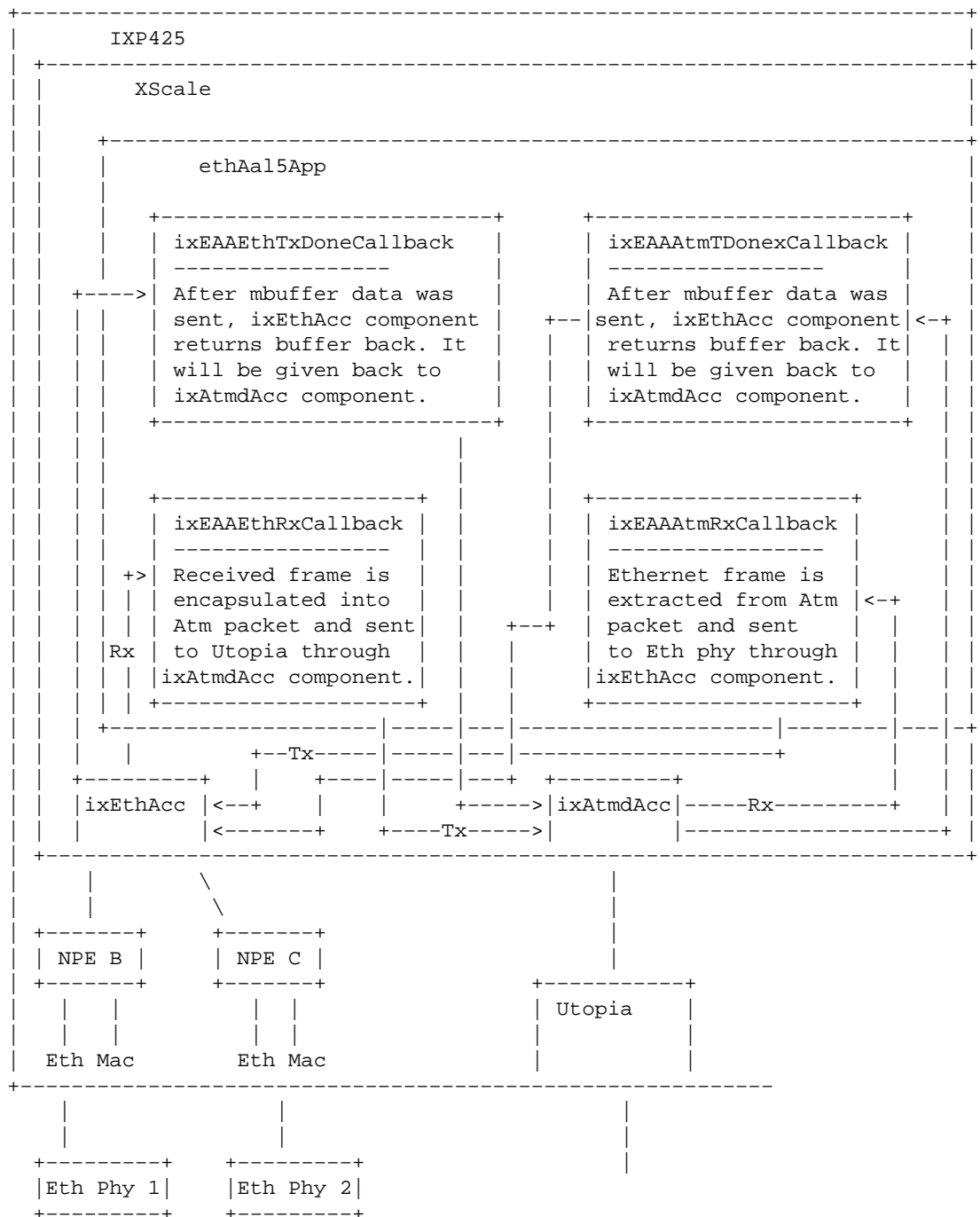
## VC Based Multiplexing of Bridged Protocols

PDU's of bridged protocols shall be carried in the Payload of the AAL5 CPCS-PDU.

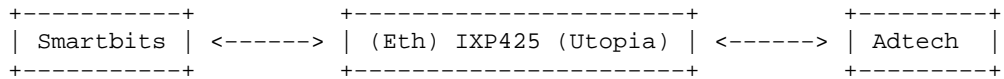




Data flow is illustrated in the diagram below:



### Configuration Example:

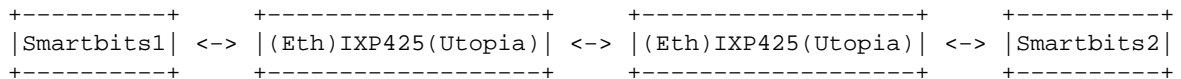


On smartbits set: dst. mac = MAC1 and src mac = MAC2

On Adtech add AAL5 with eth. encapsulation, dst. mac = MAC2, src. MAC=MAC1

Dependently which Phy is used Adtech must use same Vpi/Vci address as assigned to Phy by ethAal5App. By default for Phy 0 it is 1/100 (Vpi/Vci), for Phy 1 2/100 and so on.

Another option is:



On smartbits1 set: dst. mac = MAC1 and src mac = MAC2

On smartbits2 set: dst. mac = MAC2 and src mac = MAC1

on IXP425 connected to Smartbits 1 execute in windshell:

```
ixEAAAddMAC( 1, MAC1[0], MAC1[1], ..., MAC1[5] );
```

We use port 1 for Adsl card, however it may change in the future.

### IMPORTANT!!!

If validation board is used (instead of using IXDP425 development board), then uncomment VALIDATION\_PLATFORM\_USED flag in component.mk

---

## Define Documentation

```
#define IX_EAA_MAC1
```

Define default for MAC1 address for ixEAAAddMac() function.

Definition at line **379** of file **IxEthAal5App.h**.

```
#define IX_EAA_MAC2
```

Define default for MAC2 address for ixEAAAddMac() function.

Definition at line **385** of file **IxEthAal5App.h**.

```
#define IX_EAA_MAC3
```

Define default for MAC3 address for ixEAAAddMac() function.

Definition at line **391** of file **IxEthAal5App.h**.

```
#define IX_EAA_MAC4
```

Define default for MAC4 address for ixEAAAddMac() function.

Definition at line **397** of file **IxEthAal5App.h**.

```
#define IX_EAA_MAC5
```

Define default for MAC5 address for ixEAAAddMac() function.

Definition at line **404** of file **IxEthAal5App.h**.

```
#define IX_EAA_MAC6
```

Define default for MAC6 address for ixEAAAddMac() function.

Definition at line **410** of file **IxEthAal5App.h**.

```
#define IX_EAA_NUM_ATM_PORTS
```

Define number of supported atm ports by this application.

User can define them in **EthAal5User.h** file to overwrite definitions below

Definition at line **303** of file **IxEthAal5App.h**.

```
#define IX_EAA_NUM_BUFFERS_PER_ETH
```

This is the number of buffers, which can be stored in free buffer queue for each ethernet port.

Definition at line **288** of file **IxEthAal5App.h**.

```
#define IX_EAA_NUM_BUFFERS_PER_VC
```

This is the number of buffers per atm port.

User can define them in **EthAal5User.h** file to overwrite definitions below

Definition at line **316** of file **IxEthAal5App.h**.

```
#define IX_EAA_PORT0
```

Define default for Port number for ixEAAAddMac() function.

Definition at line **416** of file **IxEthAal5App.h**.

```
#define IX_EAA_PORT1_VPI
```

Define default VPI and VCI for 8 ports. User can define them in **EthAal5User.h** file to overwrite definitions below.

Definition at line **326** of file **IxEthAal5App.h**.

---

## Function Documentation

```
ixEthAal5AppCodeletMain ( IxEAAModeType modeType )
```

This is the main function that executes the EthAal5App codelet.

It first calls IxEAAMain() function which initialize the MAC database, to be an in valid Mac addresses (i.e. contain 0xffs), QMGR, NpeMh, Eth phys

- 100Mbit, FULL DUPLEX, NO AUTONEGOTIATION (User can change those settings accordingly to required configuration), ATM, and Utopia interface.

If Linux is used, use interrupt mode – much faster under Linux than polling

If vxWorks is used use poll mode – much faster under vxWorks than interrupts and start background QMgr queues poll

After which the main has been called, this **ixEthAal5AppCodeletMain()** function will add the MAC address using ixEAAAddMAC() function. For a single PHY utopia mode, only one port will be setup a single MAC address. However, if multiple phy is used, 8 ports will be setup and each port is assigned with a unique MAC addresses.

For single utopia phy. The following is setup using ixEAAAddMAC() function

```
ixEAAAddMAC( IX_EAA_PORT0 ,  
             IX_EAA_MAC1 ,  
             IX_EAA_MAC2 ,  
             IX_EAA_MAC3 ,  
             IX_EAA_MAC4 ,  
             IX_EAA_MAC5 ,  
             IX_EAA_MAC6 ) ;
```

For multi phy utopia the port number and MAC6 increments using a for-loop

```
for (port = 0, nextMac = 0; port < IX_EAA_NUM_ATM_PORTS; port++, nextMac++)  
{  
    ixEAAAddMAC(IX_EAA_PORT0 + port ,  
                IX_EAA_MAC1 ,  
                IX_EAA_MAC2 ,
```

```
IX_EAA_MAC3,  
IX_EAA_MAC4,  
IX_EAA_MAC5,  
IX_EAA_MAC6 + nextMac);  
}
```

Lastly, **ixEthAal5AppCodeletMain()** creates a thread which purposed to display the EthAal5App codelet counter every 15secs

# IXP425 Ethernet Access Codelet (IxEthAccCodelet) API

## [IXP425 Codelets]

IXP425 Ethernet Access Codelet API.

## Defines

```
#define IX_ETHACC_CODELET_NPEB_MAC  
    Hard-encoded MAC address for NPEB.  
  
#define IX_ETHACC_CODELET_NPEC_MAC  
    Hard-encoded MAC address for NPEC.  
  
#define IX_ETHACC_CODELET_RX_MBUF_POOL_SIZE  
    Size of receive MBuf pool.  
  
#define IX_ETHACC_CODELET_TX_MBUF_POOL_SIZE  
    Size of transmit MBuf pool.  
  
#define IX_ETHACC_CODELET_MAX_PORT  
    Number of Ethernet Ports supported for this codelet.  
  
#define IX_ETHACC_CODELET_MBUF_POOL_SIZE  
    Size of MBuf pool.  
  
#define IX_ETHACC_CODELET_PCK_SIZE  
    Size of MBuf packet (recommmaended size for ethAcc component).  
  
#define IX_ETHACC_CODELET_PCK_LEN  
    Length of MBuf payload (in bytes).  
  
#define IX_ETHACC_CODELET_MBUF_DATA_POOL_SIZE  
    Size of MBuf data pool.  
  
#define IX_ETHACC_CODELET_TXGEN_PCK_LEN  
    Size of packets for TxGenRxSink Operation.  
  
#define IX_ETHACC_CODELET_TXGEN_PCKS  
    Number of packets to generate for the TxGenRxSink Operation.  
  
#define IX_ETHACC_CODELET_RX_FCS_STRIP  
    Strip FCS from incoming frames. To undefine, change to #undef.  
  
#define IX_ETHACC_CODELET_FRAME_SIZE  
    Maximum size of a frame.
```



# Functions

PUBLIC IX\_STATUS **ixEthAccCodeletMain** (**IxEthAccCodeletOperation** operationType)

*This function is used as a single point of execution for EthAcc codelet.*

---

## Detailed Description

IXP425 Ethernet Access Codelet API.

This codelet demonstrates both Ethernet Data and Control plane services and Ethernet Management services.

- A) Ethernet Data and Control plane services:
  - ◆ Configuring both ports as a receiver sink from an external source (such as Smartbits).
  - ◆ Configuring Port–1 to automatically transmit frames and receive frames on Port–2. Frames generated and transmitted in Port–1 are loopbacked into Port–2 by using cross cable.
  - ◆ Configuring and performing a software loopback on each of the two ethernet ports.
  - ◆ Configuring both ports to act as a bridge so that frames received on one port are retransmitted on the other.
- B) Ethernet Management services:
  - ◆ Adding and removing static/dynamic entries.
  - ◆ Calling the maintenance interface (shall be run as a separate background task)
  - ◆ Calling the show routine to display the MAC address filtering tables.

### Definition

In the context of this codelet, the following definitions are applicable.

Port 1 = ixel = Ethernet port associated with NPE–B Ethernet Coprocessor.

Port 2 = ixel = Ethernet port associated with NPE–C Ethernet Coprocessor.

### Design constraints

This codelet assumes that the underlying IXP425 Product Line Silicons have two Ethernet NPEs. For silicon with single Ethernet NPE, operation will be only functional in the particular Ethernet port that corresponds to the available Ethernet NPE. Particularly, bridge operation will not work as two Ethernet ports are needed in this operation.

### Assumptions

This codelet illustrates the use EthAcc APIs. The operations provided may not be working on the best performance as the target of this codelet is just to show the functionality of APIs. In order to get better performance, #undef IX\_ETHACC\_CODELET\_TXGENRXSINK\_VERIFY to disable traffic verification.

Please note that this codelet is not optimized for production quality.

For performance testing, please use the operations below:

- Rx Sink Operation.
- TxGenRxSink Operation.

- Bridge Operation with Ethernet frames sent into either one of the Ethernet Ports.

The operations below need special tuning to optimize them. Tuning can be done by either using a lower traffic(frames/second), reducing the value of IX\_ETHACC\_CODELET\_TXGEN\_PCKS or #undef IX\_ETHACC\_CODELET\_TXGENRXSINK\_VERIFY.

- Software Loopback Operation.
- PHY Loopback Operation.
- Bridge Operation with Ethernet frames sent into both Ethernet Ports.

## VxWorks User Guide

**ixEthAccCodeletMain()** function is used as a single point of execution for EthAcc Codelet. It allows user to enter selection for different type of supported operations described below:

Usage :

```
>ixEthAccCodeletMain (operationType)
```

Where operationType:

- 1 = To sink received frames as fast as possible for available ports.
- 2 = To software loopback received frames to the same port for available ports.
- 3 = To generate and transmit frames from port 1, remote loopback by using an external cross cable to port 2, and received on port 2 (TxGenRxSink).
- 4 = To generate frames and perform PHY loopback on the same port for available ports.
- 5 = To transmit any frame received on one port through the other one (Bridge).
- 6 = To activate Ethernet MAC learning facility.

## Linux User Guide

The idea of using the **ixEthAccCodeletMain()** as a single point of execution for EthAcc codelet. The operation selected will be executed when user issue 'insmod' in command prompt.

Usage :

```
>insmod ixp400_codelets_ethAcc.o operationType=
```

Where x:

- 1 = To sink received frames as fast as possible for available ports.
- 2 = To software loopback received frames to the same port for available ports.
- 3 = To generate and transmit frames from port 1, remote loopback by using an external cross cable to port 2, and received on port 2 (TxGenRxSink).
- 4 = To generate frames and perform PHY loopback on the same port for available ports.
- 5 = To transmit any frame received on one port through the other one (Bridge).
- 6 = To activate Ethernet MAC learning facility.

## MAC Setup

The default MAC setup will be:

- Promiscuous mode enabled (for learning example)
- Frame Check Sequence appended for all frames generated on the XScale

## PHY Setup

This codelet uses two PHYs as defined by IX\_ETHACC\_CODELET\_MAX\_PHY The default PHY setup will be:

- 100Mbps,
- full duplex,
- auto-negotiation on.

### Jumbo frames

This codelet setup enable Jumbo frame reception The default setup will be:

- frames up to a msdu size of 9018 are supported.

### Test Equipment

The test harness will consist of external test equipment capable of generating Ethernet packets (e.g. SmartBits).

The test equipment must be capable of performing at least the following actions to support the scenarios outlined for the Codelet:

- Send/receive an Ethernet data stream.
- Send/receive frames of different length.
- Detect CRC errors.
- Append FCS.
- Support 100Mbit full duplex mode.

---

## Define Documentation

```
#define IX_ETHACC_CODELET_FRAME_SIZE
```

Maximum size of a frame.

This maximum frame size includes different network settings :

- Ethernet frames (up to 1518 bytes),
- BabyJumbo frames (up to nearly 1600 bytes)
- Jumbo frames (9K bytes). Note that different encapsulation types may extend the MTU size of 9000. The NPE firmware compares onlu the overall ethernet frame size (MSDU), with may be stripped from the FCS at the time of comparrison.

Definition at line **379** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_MAX_PORT
```

Number of Ethernet Ports supported for this codelet.

Definition at line **262** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_MBUF_DATA_POOL_SIZE
```

Size of MBuf data pool.

Definition at line **300** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_MBUF_POOL_SIZE
```

Size of MBuf pool.

Definition at line **271** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_NPEB_MAC
```

Hard-encoded MAC address for NPEB.

Definition at line **226** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_NPEC_MAC
```

Hard-encoded MAC address for NPEC.

Definition at line **235** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_PCK_LEN
```

Length of MBuf payload (in bytes).

Definition at line **291** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_PCK_SIZE
```

Size of MBuf packet (recommended size for ethAcc component).

Definition at line **282** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_RX_FCS_STRIP
```

Strip FCS from incoming frames. To undefine, change to #undef.

```
#define IX_ETHACC_CODELET_RX_MBUF_POOL_SIZE
```

Size of receive MBuf pool.

Definition at line **244** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_TX_MBUF_POOL_SIZE
```

Size of transmit MBuf pool.

Definition at line **253** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_TXGEN_PCK_LEN
```

Size of packets for TxGenRxSink Operation.

Definition at line **310** of file **IxEthAccCodelet.h**.

```
#define IX_ETHACC_CODELET_TXGEN_PCKS
```

Number of packets to generate for the TxGenRxSink Operation.

Definition at line **319** of file **IxEthAccCodelet.h**.

---

## Function Documentation

```
ixEthAccCodeletMain ( IxEthAccCodeletOperation operationType )
```

This function is used as a single point of execution for EthAcc codelet.

### **Parameters:**

*IxEthAccCodeletOperation* [in]  
*operationType*

– The type of operation to be executed. Refer to the descriptions above.

### **Returns:**

- ◇ IX\_SUCCESS : If operation selected is successfully setup
- ◇ IX\_FAIL : If operation selected fails to be setup.

# IXP425 HSS Access Codelet (IxHssAccCodelet) API

## [IXP425 Codelets]

IXP425 HSS Access Codelet API. The interface for the HSS Access Codelet.

## Defines

```
#define IX_HSSACC_CODELET_DURATION_IN_MS
```

## Functions

```
PUBLIC ixHssAccCodeletMain (IxHssAccCodeletOperation operationType,  
IX_STATUS IxHssAccCodeletPortMode portMode, IxHssAccCodeletVerifyMode verifyMode)
```

---

## Detailed Description

IXP425 HSS Access Codelet API. The interface for the HSS Access Codelet.

This module contains the implementation of the HSS Access Codelet.

The following top-level operation is supported:

- Test Packetised and Channelised Services, with the Codelet acting as data source/sink and HSS as loopback. The Codelet will transmit data and verify that data received is the same as that transmitted. Codelet runs for IX\_HSS\_CODELET\_DURATION\_IN\_MS ms.
- There are four clients of Packetised service per HSS port – 1 client per E1/T1 trunk. Client 0 and 2 are running in Packetised HDLC mode while client 1 and 3 in Packetised RAW mode.

### Assumptions

In Channelised service, the codelet transmits traffic continuously. When the codelet runs up to IX\_HSS\_CODELET\_DURATION\_IN\_MS ms, Tx counter is bigger than Rx counter. This is due to the fact that traffics submitted to NPE (i.e. Tx counter has been increased) are not transmitted out by NPE when HSS service is disabled. These traffics will be dropped and not loopbacked at HSS (Hence, Rx counter not increased).

In Packetised-raw mode service (client 1 and 3), Rx counter will be bigger than Tx counter because in this service, idle packets are received by XScale and causes Rx counter to be bigger than Tx counter. As for packetised-HDLC service, idle packets are handled in HDLC coprocessor and not passed to XScale (Hence, Rx counter not increased).

### Limitations

When executing Packetised service on both HSS ports of 266MHz IXP421 simultaneously, receive traffic verification should fail on client 3 (i.e. Packetised-raw mode) of HSS port 1. The reason why this issue

occured is IXP421 does not have enough CPU resources to perform intensive packet verification tasks. However, this does not imply that the same issue will hit customer applications because actual applications will not do any packet verifications in the way that the codelet does. This issue is not seen on a 533MHz IXP425.

## VxWorks User Guide

**ixHssAccCodeletMain()** function is used as a single point of execution for HssAcc Codelet.

```
Usage :
>ixHssAccCodeletMain (operationType, portMode, verifyMode)
Where operationType:
    1 = Packetised Service Only.
    2 = Channelised Service Only.
    3 = Packetised Service and Channelised Services.

Where portMode:
    1 = HSS Port 0 Only.
    2 = HSS Port 1 Only.
    3 = HSS Port 0 and 1.

Where verifyMode:
    1 = codelet verifies traffic received in hardware loopback mode.
    2 = codelet does not verify traffic received in hardware loopback mode.
```

## Linux User Guide

The idea of using the **ixHssAccCodeletMain()** as a single point of execution for HssAcc codelet. The operation selected will be executed when user issue 'insmod' in command prompt.

```
Usage :
>insmod ixp400_codelets_hssAcc.o operationType=(a) portMode=(b) verifyMode=(c)
Where a:
    1 = Packetised Service Only.
    2 = Channelised Service Only.
    3 = Packetised Service and Channelised Services.

Where b:
    1 = HSS Port 0 Only.
    2 = HSS Port 1 Only.
    3 = HSS Port 0 and 1.

Where c:
    1 = codelet verifies traffic received in hardware loopback mode.
    2 = codelet does not verify traffic received in hardware loopback mode.
```

## Buffer Management

The packetised service uses mbuf buffers to store data, and chains mbufs together to form large packets. In the transmit direction, mbufs are allocated from a pool on transmit, and returned to the pool on transmit done. For receive, mbufs are allocated from a pool when supplying buffers to the free queue, and returned to the pool on receive.

The channelised service operates quite differently. As voice data is very sensitive to latency using mbufs for transferral of the data between XScale and NPE is not very appropriate. Instead, circular buffers are used whereby the NPE reads data from a block of SDRAM that the XScale writes to, and writes data to a block of SDRAM that the XScale reads from. On receive, the NPE writes directly into a circular buffer that the XScale subsequently reads the data from. Each channel has its own circular buffer, and all these buffers are stored contiguously. On transmit, the NPE takes a circular list of pointers from the XScale and transmits the data referenced by these pointers. Each list of pointers contains a pointer for each channel, and the circular list of pointers contains multiple lists stored contiguously. This is to allow the XScale to transmit voice samples without having to copy data, as only the pointer to the data blocks needs to be written to SDRAM. The NPE lets the XScale know, in the form of Tx and Rx offsets, where in the blocks of SDRAM it is currently reading from and writing to. This enables the XScale to co-ordinate its reading and writing activities to maintain the data flow. The Tx offset lets the XScale know the list offset into the Tx circular pointer list that the NPE will next use to transmit. The Rx offset lets the XScale know the byte offset into each channel's Rx circular buffer that the NPE will next receive data into.

## **Caching**

To improve system performance, caching may be enabled for both the channelised and packetised buffers. To allow for this, buffers need to be flushed before transmit, and invalidated after receive. Flushing the buffers before transmit ensures the NPE reads and transmits the correct data. Invalidating the buffers after receive ensures the XScale reads and processes the correct data. In the case of the Codelet, all data is flushed and invalidated as every byte is being written to on transmit and every byte is verified on receive. In a real application flushing or invalidating all the data may not be necessary, only the data that the application has written before transmit or will read after receive. Note, regarding the packetised service, the IxHssAcc component itself takes care of flushing and invalidating mbuf headers. The application needs only to concern itself with the mbuf data.

## **Data Verification Strategy**

For both the packetised and channelised service a changing pattern will be transmitted. When the HSS co-processor is performing a loopback the data received is expected to be the same as that transmitted. The data transmitted carries a byte pattern that begins at a known value and is incremented for each byte. An independent byte pattern is transmitted for each channel of the channelised service, and also for each port of the packetised service. When data is received it is expected to match the pattern that was transmitted. For the channelised service the first non-idle byte received is expected to be the beginning of the byte pattern. For the packetised service, RAW mode clients may receive idle data so this is detected and ignored. Only non-idle data is verified.

## **56Kbps Packetised HDLC feature**

This feature is demonstrated in one of the packetised HDLC clients (i.e. client 2). The CAS bit is configured to be in the least significant bit (LSB) position with bit polarity '1'. Bit inversion is also enabled on this client as well. The data verification strategy remains the same as other packetised clients.



# IXP425 PerfProf Access Codelet

## [IXP425 Codelets]

(IxPerfProfAccCodelet) API IXP425 codelet PerfProf Access component API

## Defines

```
#define PSS_MASK
    Masks out PSS portion of the PMSR register.
```

```
#define EXPANSION_BUS
#define SDRAM_CONTROLLER
#define PCI
#define QUEUE_MANAGER
#define AHB_APB_BRIDGE
```

## Enumerations

```
enum IxPerfProfAccCodeletMode {
    IX_PERFPROF_ACC_CODELET_MODE_HELP,
    IX_PERFPROF_ACC_CODELET_MODE_ALL,
    IX_PERFPROF_ACC_CODELET_MODE_BUS_PMU_NORTH_MODE,
    IX_PERFPROF_ACC_CODELET_MODE_BUS_PMU_SOUTH_MODE,
    IX_PERFPROF_ACC_CODELET_MODE_BUS_PMU_SDRAM_MODE,
    IX_PERFPROF_ACC_CODELET_MODE_XSCALE_PMU_EVENT_SAMPLING,
    IX_PERFPROF_ACC_CODELET_MODE_XSCALE_PMU_TIME_SAMPLING,
    IX_PERFPROF_ACC_CODELET_MODE_XSCALE_PMU_EVENT_COUNTING,
    IX_PERFPROF_ACC_CODELET_MODE_XCYCLE,
    IX_PERFPROF_ACC_CODELET_MODE_BUS_PMU_PMSR_GET
}
    Contains selection of mode to be used when calling the main API.
```

## Functions

```
PUBLIC IxPerfProfAccCodeletMain (IxPerfProfAccCodeletMode mode, UINT32 param1, UINT32
    void param2, UINT32 param3, UINT32 param4, UINT32 param5, UINT32 param6, UINT32 param7,
    UINT32 param8, UINT32 param9)
```

---

## Detailed Description

(IxPerfProfAccCodelet) API IXP425 codelet PerfProf Access component API

Functionality of the PerfProf Access Codelet

The codelet shall demonstrate how the Performance Profiling utility can be used for profiling purposes.

- The different implementations are demonstrated.
  - ◆ Help – Lists down how the codelet can be used.
  - ◆ Demo All – A non user configurable demonstration on how to use the APIs
  - ◆ Bus Pmu North Mode – Profiling of the north bus activities. Enables user to select events that they wish to monitor.
  - ◆ Bus Pmu South Mode – Profiling of the south bus activities. Enables user to select events that they wish to monitor.
  - ◆ Bus Pmu Sdram Mode – Profiling of the sdram bus activities. Enables user to select events that they wish to monitor.
  - ◆ Bus Pmu PMSR Get Mode – Get the last slave or master to access the bus.
  - ◆ Xscale PMU Event Sampling – Event Sampling of Xscale PMU. Enables user to select event and sampling rate that they wish to sample.
  - ◆ Xscale PMU Time Sampling – Time Sampling of Xscale PMU. Enables user to select clock count mode and number of events and rate they wish to sample.
  - ◆ Xscale PMU Event Counting – Event counting of Xscale PMU. Enables users to select events that they wish to count or monitor.
  - ◆ Xcycle Measurement – Measurement of cycle idle time. i.e when the cycles are not being used to process anything.

## User Guide

Users will be able to start the codelet by calling ixPerfProfAccCodeletMain and passing in up to 10 parameters. The parameters are represented in the following order:

### **Help Mode**

*Mode - Select IX\_PERFPROF\_ACC\_CODELET\_MODE\_HELP  
Set the rest of the parameters to 0.*

### **All functionalities mode**

*Mode - Select IX\_PERFPROF\_ACC\_CODELET\_MODE\_ALL  
Set the rest of the parameters to 0.*

### **Bus PMU north mode**

*Mode - Select IX\_PERFPROF\_ACC\_CODELET\_MODE\_BUS\_PMU\_NORTH\_MODE  
param1 - Select proper PEC1 value from main header file.  
param2 - Select proper PEC2 value from main header file.  
param3 - Select proper PEC3 value from main header file.  
param4 - Select proper PEC4 value from main header file.  
param5 - Select proper PEC5 value from main header file.  
param6 - Select proper PEC6 value from main header file.  
param7 - Select proper PEC7 value from main header file.  
Set the rest of the parameters to 0.*

### **Bus PMU south mode**

*Mode - Select IX\_PERFPROF\_ACC\_CODELET\_MODE\_BUS\_PMU\_SOUTH\_MODE  
param1 - Select proper PEC1 value from main header file.  
param2 - Select proper PEC2 value from main header file.  
param3 - Select proper PEC3 value from main header file.  
param4 - Select proper PEC4 value from main header file.  
param5 - Select proper PEC5 value from main header file.  
param6 - Select proper PEC6 value from main header file.  
param7 - Select proper PEC7 value from main header file.  
Set the rest of the parameters to 0.*

### **Bus PMU sdram mode**

*Mode - Select IX\_PERFPROF\_ACC\_CODELET\_MODE\_BUS\_PMU\_SDRAM\_MODE  
param1 - Select proper PEC1 value from main header file.  
param2 - Select proper PEC2 value from main header file.  
param3 - Select proper PEC3 value from main header file.*

param4 - Select proper PEC4 value from main header file.  
 param5 - Select proper PEC5 value from main header file.  
 param6 - Select proper PEC6 value from main header file.  
 param7 - Select proper PEC7 value from main header file.  
 Set the rest of the parameters to 0.

**Bus PMU PMSR Get**

Mode - Select IX\_PERFPROF\_ACC\_CODELET\_MODE\_BUS\_PMU\_PMSR\_GET  
 Set the rest of the parameters to 0.

**XScale PMU Event Sampling**

Mode - Select IX\_PERFPROF\_ACC\_CODELET\_MODE\_DEMO\_XSCALE\_PMU\_EVENT\_SAMPLING  
 param1 - Number of events  
 param2 - Event 1  
 param3 - Sampling rate of Event 1  
 param4 - Event 2  
 param5 - Sampling rate of Event 2  
 param6 - Event 3  
 param7 - Sampling rate of Event 3  
 param8 - Event 4  
 param9 - Sampling rate of Event 4

**XScale PMU Time Sampling**

Mode - Select IX\_PERFPROF\_ACC\_CODELET\_MODE\_DEMO\_XSCALE\_PMU\_TIME\_SAMPLING  
 param1 - Sampling rate.  
 param2 - Clock count divider.  
 Set the rest of the parameters to 0.

**XScale PMU Event Counting**

Mode - Select IX\_PERFPROF\_ACC\_CODELET\_MODE\_DEMO\_XSCALE\_PMU\_EVENT\_COUNTING  
 param1 - Clock count divider.  
 param2 - Number of events.  
 param3 - Event 1.  
 param4 - Event 2.  
 param5 - Event 3.  
 param6 - Event 4.  
 Set the rest of the parameters to 0.

**Xcycle Measurement**

Mode - Select IX\_PERFPROF\_ACC\_CODELET\_MODE\_XCYCLE  
 param1 - Number of runs required.  
 Set the rest of the parameters to 0.

## VxWorks User Guide

**ixPerfProfAccCodeletMain()** function is used as a single point of execution for PerfProfAcc Codelet. It allows user to enter selection for different type of supported operations as described above.

Usage :

```
>ixPerfProfAccCodeletMain (mode, param1, param2, param3, param4, param5, param6,
                             param7, param8, param9)
```

Where mode and params are described above.

## Linux User Guide

**ixPerfProfAccCodeletMain()** function is used as a single point of execution for PerfProfAcc Codelet. It allows user to enter selection for different type of supported operations as described above.

Usage :

```
>insmod ixp400_codelets_perfProfAcc.o \
mode= \
param1= \
param2= \
```

```

param3= \
param4= \
param5= \
param6= \
param7= \
param8= \
param9= \

```

Where Parameter X are as described above.

---

## Define Documentation

```
#define PSS_MASK
```

Masks out PSS portion of the PMSR register.

Definition at line **230** of file **IxPerfProfAccCodelet.h**.

---

## Enumeration Type Documentation

```
enum IxPerfProfAccCodeletMode
```

Contains selection of mode to be used when calling the main API.

### ***Enumeration values:***

<i>IX_PERFPROF_ACC_CODELET_MODE_HELP</i>	Select help mode.
<i>IX_PERFPROF_ACC_CODELET_MODE_ALL</i>	Select all mode.
<i>IX_PERFPROF_ACC_CODELET_MODE_BUS_PMU_NORTH_MODE</i>	Select north bus pmu mode.
<i>IX_PERFPROF_ACC_CODELET_MODE_BUS_PMU_SOUTH_MODE</i>	Select south bus pmu mode.
<i>IX_PERFPROF_ACC_CODELET_MODE_BUS_PMU_SDRAM_MODE</i>	Select sdram mode.
<i>IX_PERFPROF_ACC_CODELET_MODE_XSCALE_PMU_EVENT_SAMPLING</i>	Select xscale pmu event sampling mode.
<i>IX_PERFPROF_ACC_CODELET_MODE_XSCALE_PMU_TIME_SAMPLING</i>	Select xscale pmu time sampling mode.
<i>IX_PERFPROF_ACC_CODELET_MODE_XSCALE_PMU_EVENT_COUNTING</i>	Select xscale pmu event counting mode.
<i>IX_PERFPROF_ACC_CODELET_MODE_XCYCLE</i>	Select xcycle mode.
<i>IX_PERFPROF_ACC_CODELET_MODE_BUS_PMU_PMSR_GET</i>	

Select bus pmu  
pmsr get mode.

Definition at line **289** of file **IxPerfProfAccCodelet.h**.

# IXP425 Timers (IxTimersCodelet) API

## [IXP425 Codelets]

IXP425 Timer component API.

## Typedefs

```
typedef void(* IxTimerIsr )(void *arg)  
Timer callback prototype.
```

## Enumerations

```
enum IxTimerId {  
    IX_TIMER_1,  
    IX_TIMER_2,  
    IX_TIMER_WDOG,  
    IX_TIMER_TS,  
    IX_TIMER_PMU,  
    IX_TIMER_MAX  
}  
Hardware timers.
```

## Functions

```
PUBLIC IX_STATUS ixTimerInit (BOOL recoverFromLostStatus)  
Initialise the Timer API.
```

```
PUBLIC IX_STATUS ixTimerBind (IxTimerId timer, IxTimerIsr isr, void *arg)  
Initialise a Timer.
```

```
PUBLIC IX_STATUS ixTimerEnable (IxTimerId timer, UINT32 downCounter, BOOL oneShot)  
Enable a Timer.
```

```
PUBLIC IX_STATUS ixTimerDisable (IxTimerId timer)  
Disable a Timer.
```

```
PUBLIC IX_STATUS ixTimerDownCounterGet (IxTimerId timer, UINT32 *downCounter)  
Get the remaining time of a timer.
```

```
PUBLIC IX_STATUS ixTimerUnbind (IxTimerId timer)  
Unregister a Timer.
```

```
PUBLIC void ixTimerShow (void)  
Show internal counters.
```

PUBLIC IX\_STATUS **ixTimerMemMap** (void)  
*Maps the timer address space.*

PUBLIC void **ixTimerMemUnmap** (void)  
*Unmaps the timer address space.*

---

## Detailed Description

IXP425 Timer component API.

These utilities provide support for enabling, triggering and disabling timers

---

## Typedef Documentation

**IxTimerIsr**

Timer callback prototype.

Definition at line **105** of file **IxTimersCodelet.h**.

---

## Enumeration Type Documentation

**enum IxTimerId**

Hardware timers.

**Enumeration values:**

<b><i>IX_TIMER_1</i></b>	General Purpose Timer 1.
<b><i>IX_TIMER_2</i></b>	General Purpose Timer 2.
<b><i>IX_TIMER_WDOG</i></b>	Watchdog Timer.
<b><i>IX_TIMER_TS</i></b>	Timestamp Timer.
<b><i>IX_TIMER_PMU</i></b>	XScale core PMU timer.
<b><i>IX_TIMER_MAX</i></b>	Delimiter for error checking.

Definition at line **88** of file **IxTimersCodelet.h**.

# Function Documentation

```
ixTimerBind ( IxTimerId timer,  
              IxTimerIsr isr,  
              void *    arg  
            )
```

Initialise a Timer.

This function is called to initialise one of the timers

**Parameters:**

*timer* (in) the timer to initialise  
*isr* (in) the callback to register  
*arg* (in) the parameter to be passed to the callback

**Returns:**

◇ IX\_SUCCESS, the timer successfully initialised  
  
◇ IX\_FAIL, failed to initialize the timer

```
ixTimerDisable ( IxTimerId timer )
```

Disable a Timer.

This function is called to disable a timer

**Parameters:**

*timer* (in) the timer to disable

**Returns:**

◇ IX\_SUCCESS, the timer successfully disabled  
  
◇ IX\_FAIL, failed to disable the timer

**Note:**

This function can be used from an interrupt context

```
ixTimerDownCounterGet ( IxTimerId timer,  
                        UINT32 * downCounter  
                      )
```



Get the remaining time of a timer.

This function is called to get the remaining time before the next interrupt triggers

If this function is used after a timer expires, the result is a negative value. Its unsigned representation is a number close to 0xffff....

**Note:**

Watchdog timer stops when expiring. Then the counter retrieved by this function is always 0.

**Parameters:**

*timer* (in) the timer to query  
*downCounter* (out) the timer time left

**Returns:**

- ◇ IX\_SUCCESS, the timer successfully queried
- ◇ IX\_FAIL, failed to query the timer

**Note:**

This function can be used from an interrupt context

```
ixTimerEnable ( IxTimerId timer,  
                UINT32   downCounter,  
                BOOL     oneShot  
                )
```

Enable a Timer.

This function is called to enable a timer

**Parameters:**

*timer* (in) the timer to enable  
*downCounter* (in) the number of cycles between interrupts. downCounter should be a multiple of 4 for *IX\_TIMER\_1* and *IX\_TIMER\_2* timers. This counter is in pClock cycles (peripheral cycles) except for the *IX\_TIMER\_PMU* where the unit is in xClock cycles (processor frequency)  
*oneShot* (in) the type of timer to trigger. When set to TRUE, the interrupt will fire once when the downCounter reaches 0. When set to FALSE, the interrupts will fire every downCounter bus cycles. FALSE is not supported for *IX\_TIMER\_WDOG* and *IX\_TIMER\_TS* timers.

**Returns:**

- ◇ IX\_SUCCESS, the timer successfully enabled
- ◇ IX\_FAIL, failed to enable the timer

**Note:**

This function can be used from an interrupt context

**ixTimerInit ( BOOL *recoverFromLostStatus* )**

Initialise the Timer API.

This function must be called before any other IxTimer function. It sets up internal data structures.

**Parameters:**

*recoverFromLostStatus* (in) set to true to enable Lost Status workaround

**Returns:**

◇ IX\_SUCCESS, the IxTimer API successfully initialised

◇ IX\_FAIL, failed to initialize the IxTimer API

**ixTimerMemMap ( void )**

Maps the timer address space.

This function is called in order to map the physical address space for the timer to virtual address space

**Parameters:**

*none*

**Returns:**

◇ IX\_SUCCESS, the timer physical address space was mapped to virtual address space

◇ IX\_FAIL, the timer physical address space failed to be mapped to virtual address space

**ixTimerMemUnmap ( void )**

Unmaps the timer address space.

This function is called in order to unmap the physical address space for the timer

**Parameters:**

*none*

**Returns:**

*none*

**ixTimerShow ( void )**

Show internal counters.

This function is called to show the internal counters incremented when Lost Status workaround is needed.

**Parameters:**

*none*

**Returns:**

◇ void

```
ixTimerUnbind ( IXTimerId timer )
```

Unregister a Timer.

This function is called to unregister a timer

**Parameters:**

*timer* (in) the timer to unregister

**Returns:**

◇ IX\_SUCCESS, the timer  
successfully removed

◇ IX\_FAIL, failed to remove the  
timer

# IXP425 USB RNDIS Codelet (IxUSBRNDIS) API

## [IXP425 Codelets]

IXP425 codelet USB RNDIS API.

## Modules

**IXP425 USB RNDIS End Driver Codelet (IxUSBRNDIS) API**

*IXP425 codelet for RNDIS End API.*

**IXP425 USB RNDIS Vendor Codelet (IxUSBRNDIS)**

*IXP425 codelet for RNDIS Vendor information.*

## Functions

**PUBLIC IX\_STATUS ixUSBRNDISSignalEncapsulatedCommand** (void) *Function prototype for signalling encapsulation command.*

**PUBLIC IX\_STATUS ixUSBRNDISProcessEncapsulatedCommand** (IX\_USB\_MBLK \*) *Function prototype for processing encapsulation command.*

**PUBLIC IX\_STATUS ixUSBRNDISProcessDataPacket** (IX\_USB\_MBLK \*) *Function prototype for processing data packet.*

**PUBLIC IX\_STATUS ixUSBRNDISLayerInit** (void \*pDrvCtrl) *Function prototype for layer initialization.*

**PUBLIC IX\_STATUS ixUSBRNDISInit** (void) *Function prototype for initializing RNDIS.*

**PUBLIC void ixUSBRNDISUnload** (void) *Function prototype for releasing the I/O memory and disconnecting the interrupt.*

**PUBLIC IX\_USB\_MBLK \* ixUSBRNDISCreateMBuf** (UINT8 \*buffer, UINT32 len) *Function prototype for creating MBufs.*

**PUBLIC IX\_STATUS ixUSBRNDISSendDataPacket** (RNDIS\_BUF \*packet) *Function prototype for sending data packet. It is the hook for the RNDIS END.*

**void ixUSBRNDISIpHdrDump** (const char \*const mData) **const char \* ixUSBRNDISIpProtoStrGet** (const UINT8 ipProto) **const char \* ixUSBRNDISEthernetTypeStrGet** (const UINT16 etherType)  
**void ixUSBRNDISEthernetHdrDump** (const char \*const mData, BOOL \*nonIpHdrDetected)

---

## Detailed Description

IXP425 codelet USB RNDIS API.

How to use the USB RNDIS codelet:

- build a loadable object and load it into vxWorks
- start the codelet by typing ixUSBRNDISStart

You should see the "usb" network interface in the output generated by the ifShow command.

Plug the board into the USB port of a Windows 98/ME/2000 machine and selected the driver provided with the codelet when queried for it.

Note:

- the IP and MAC addresses of the END driver (therefore the board side of the link) are defined in ixUSBRNDISEnd.h
- the MAC address of the RNDIS driver (which will be used by Windows as its own MAC address) is defined in ixUSBRNDIS.h

Currently the END MAC address is 00:00:00:00:00:02 and the RNDIS MAC address is 00:00:00:00:00:01. The IP address of the END is 192.168.1.1, therefore you should use a compatible address for the RNDIS controller on the Windows side (such as 192.168.1.2) and set the END IP address as gateway address for the RNDIS network device, or change them to suitable values.

The codelet was tested with Windows 2000 only, and telnet and ftp traffic was passed by routing the PC through the IXP425 into a network.

Warning: this codelet is for demonstration purposes only, it should not be considered a fully working application.

---

## Function Documentation

```
PUBLIC IX_USB_MBLK * ixUSBRNDISCreateMBuf ( UINT8 * buffer,
                                           UINT32 len
                                           )
```

Function prototype for creating MBufs.

**Parameters:**

UINT8 \*buffer – Pointer to a buffer  
UINT32 len – Length of buffer

**Returns:**

◇ IX\_USB\_MBLK –  
Successfully create MBUF

◇ NULL – Failed to create  
MBUF

```
PUBLIC IX_STATUS ixUSBNDISInit ( void )
```

Function prototype for initializing RNDIS.

**Parameters:**

*None*

**Returns:**

◇ IX\_SUCCESS – Successfully  
initialized RNDIS

◇ IX\_FAIL – Failed to initialize  
RNDIS

```
PUBLIC IX_STATUS ixUSBNDISLayerInit ( void * pDrvCtrl )
```

Function prototype for layer initialization.

**Parameters:**

*pDrvCtrl* – Pointer to the device to be  
initialized

**Returns:**

◇ IX\_SUCCESS – Successfully  
initialized

◇ IX\_FAIL – Failed to initialize

```
PUBLIC IX_STATUS ixUSBNDISProcessDataPacket ( IX_USB_MBLK * )
```

Function prototype for processing data packet.

**Parameters:**

*IX\_USB\_MBLK* – data packet to be  
processed

**Returns:**

◇ IX\_SUCCESS – Data packet  
successfully processed

◇ IX\_FAIL – Failed to process data  
packet

```
PUBLIC IX_STATUS ixUSBNDISProcessEncapsulatedCommand ( IX_USB_MBLK * )
```

Function prototype for processing encapsulation command.

**Parameters:**

*IX\_USB\_MBLK* – memory buffer to be encapsulated

**Returns:**

◇ IX\_SUCCESS – Successfully processing encapsulation command

◇ IX\_FAIL – Failed to process encapsulation command for some internal reason

```
PUBLIC IX_STATUS ixUSBNDISSendDataPacket ( RNDIS_BUF * packet )
```

Function prototype for sending data packet. It is the hook for the RNDIS END.

**Parameters:**

*RNDIS \*packet* – Pointer to a data packet to be transmitted

**Returns:**

◇ IX\_SUCCESS – Successfully send a data packet

◇ IX\_FAIL – Failed to send a data packet

```
PUBLIC IX_STATUS ixUSBNDISSignalEncapsulatedCommand ( void )
```

Function prototype for signalling encapsulation command.

**Parameters:**

*None*

**Returns:**

◇ IX\_SUCCESS – Successfully signalling encapsulation command

◇ IX\_FAIL – Failed to signal encapsulation command for some internal reason

```
PUBLIC void ixUSBNDISUnload ( void )
```

Function prototype for releasing the I/O memory and disconnecting the interrupt.

**Parameters:**

*None*

***Returns:***

None



# IXP425 USB RNDIS End Driver Codelet (IxUSBRNDIS) API

## [IXP425 USB RNDIS Codelet (IxUSBRNDIS) API]

IXP425 codelet for RNDIS End API.

### Defines

```
#define RNDIS_END_INET_ADDR
    RNDIS END driver with inet address.

#define RNDIS_END_MAC_ADDRESS
    RNDIS END driver with MAC address.
```

### Functions

```
PUBLIC STATUS ixUSBRNDISStart (void)
    Starts the RNDIS component.

PUBLIC void rndisInt (void *pDrvCtrl, int packet)
    RNDIS END hook for receiving packets.
```

---

### Detailed Description

IXP425 codelet for RNDIS End API.

Description of the RNDIS End driver

#### Running the codelet in Linux.

- insmod lib/linuxbe/ixp400.o
- insmod lib/linuxbe/ixp400\_codelets\_usb.o
- ifconfig usb0 x.x.x.x (Where x.x.x.x is the desired ip address of the interface)

#### Running the codelet in VxWorks big endian.

- Load module that was built
  - ◆ ld < lib/vxbe/codelets\_usbTest.out
- Run the codelet
  - ◆ ixUSBRNDISStart
  - ◆ ifAddrSet "usb0", "x.x.x.x" (Where x.x.x.x is the desired ip address of the interface)

## Running the codelet in VxWorks little endian.

- Load module that was built
  - ◆ `ld < lib/vxle/codelets_usbTest.out`
- Run the codelet
  - ◆ `ixUSBNDISStart`
  - ◆ `ifAddrSet "usb0", "x.x.x.x"` (Where x.x.x.x is the desired ip address of the interface)

The codelet is executed by calling the `ixUSBNDISStart` function. No parameters are needed to be passed in. The start function first loads and initializes the device driver. It then goes on to load the device driver into the MUX. The function then goes on to call `muxDevStart()` passing in the called earlier. `muxDevStart` start the device that has already been initialized and handles registering the driver's interrupt service routine and anything else necessary to handle receiving and transmitting. The function finally assigns an IP address to the RNDIS interface.

## Overview of `IxRNDISEnd.c`

### Description of macros.

In this example driver the macros `RNDIS_OUT_SHORT` and `RNDIS_IN_SHORT` are sample macros to read/write data to a mock device. If a device communicates through formatted control blocks in shared memory, the accesses to those control blocks should also be through redefinable macros.

The macros `SYS_INT_CONNECT`, `SYS_INT_DISCONNECT`, and `SYS_INT_ENABLE` allow the driver to be customized for BSPs that use special versions of these routines.

The macro `SYS_INT_CONNECT` is used to connect the interrupt handler to the appropriate vector. By default it is the routine `intConnect()`.

The macro `SYS_INT_DISCONNECT` is used to disconnect the interrupt handler prior to unloading the module. By default this is a dummy routine that returns OK.

The macro `SYS_INT_ENABLE` is used to enable the interrupt level for the end device. It is called once during initialization. By default this is the routine `sysLanIntEnable()`, defined in the module `sysLib.o`.

The macro `SYS_ENET_ADDR_GET` is used to get the ethernet address (MAC) for the device. The single argument to this routine is the `END_DEVICE` pointer. By default this routine copies the ethernet address stored in the global variable `sysTemplateEnetAddr` into the `END_DEVICE` structure.

### Brief overview of the APIs. `ixUSBNDISStart()`

- Initializes and starts the end driver. Called to run the codelet.

`END_OBJ* rdisLoad ( char* initString, void *unused )`

- This routine initializes the driver and the device to the operational state. All of the device specific parameters are passed in the `initString`.

`rdisParse ( END_DEVICE * pDrvCtrl, char * initString )`

- Parse the input string. Fill in values in the driver control structure.

rndisMemInit ( END\_DEVICE \* pDrvCtrl )

- Initialize memory for the chip.

rndisStart ( END\_OBJ \* pDrvCtrl )

- Handle controller interrupt.

rndisRecv ( END\_DEVICE \*pDrvCtrl, M\_BLK\_ID packet )

- Process the next incoming packet.

rndisHandleRcvInt ( END\_DEVICE \*pDrvCtrl, M\_BLK\_ID packet )

- Task level interrupt service for input packets.

rndisSend ( END\_DEVICE \* pDrvCtrl, M\_BLK\_ID pMblk )

- The driver send routine.

rndisIoctl ( END\_DEVICE \* pDrvCtrl, int cmd, caddr\_t data )

- The driver I/O control routine.

rndisConfig ( END\_DEVICE \*pDrvCtrl )

- Reconfigure the interface under us.

rndisAddrFilterSet ( END\_DEVICE \*pDrvCtrl )

- Set the address filter for multicast addresses.

rndisPollRcv ( END\_DEVICE \* pDrvCtrl, M\_BLK\_ID pMblk )

- Routine to receive a packet in polled mode.

rndisPollSend ( END\_DEVICE\* pDrvCtrl, M\_BLK\_ID pMblk )

- Routine to send a packet in polled mode.

rndisMCastAdd ( END\_DEVICE \*pDrvCtrl char\* pAddress )

- Add a multicast address for the device.

rndisMCastDel ( END\_DEVICE \*pDrvCtrl, char\* pAddress )

- Delete a multicast address for the device.

rndisMCastGet ( END\_DEVICE \*pDrvCtrl, MULTI\_TABLE\* pTable )

- Get the multicast address list for the device.

rndisStop ( END\_DEVICE \*pDrvCtrl )

- Stop the device.

rndisUnload ( END\_DEVICE\* pDrvCtrl )

- Unload a driver from the system.

rndisPollStart ( END\_DEVICE \* pDrvCtrl )

- Start polled mode operations.

rndisPollStop ( END\_DEVICE \* pDrvCtrl )

- Stop polled mode operations.

rndisReset ( END\_DEVICE \*pDrvCtrl )

- Reset device.

---

## Define Documentation

```
#define RNDIS_END_INET_ADDR
```

RNDIS END driver with inet address.

Definition at line **260** of file **IxUSBRRNDISEnd.h**.

```
#define RNDIS_END_MAC_ADDRESS
```

RNDIS END driver with MAC address.

Definition at line **267** of file **IxUSBRRNDISEnd.h**.

---

## Function Documentation

```
PUBLIC STATUS ixUSBRRNDISStart ( void )
```

Starts the RNDIS component.

***Parameters:***

*None*

***Returns:***

◇ OK – Succesfully start

RNDIS component

◇ ERROR – Failed to start  
RNDIS component

```
PUBLIC void rndisInt ( void * pDrvCtrl,  
                     int    packet  
                     )
```

RNDIS END hook for receiving packets.

***Parameters:***

*pDrvCtrl* – pointer to END  
device

*packet* – received packet

***Returns:***

None

# IXP425 USB RNDIS Vendor Codelet (IxUSBRNDIS)

## [IXP425 USB RNDIS Codelet (IxUSBRNDIS) API]

IXP425 codelet for RNDIS Vendor information.

### Defines

```
#define RNDIS_VENDOR_ID
    RNDIS with static Intel Vendor ID.

#define RNDIS_PRODUCT_ID
    RNDIS with static Product ID.

#define RNDIS_VENDOR_DESCRIPTION
    RNDIS with vendor description.

#define RNDIS_MAC_ADDRESS
    RNDIS with MAC address.
```

---

### Detailed Description

IXP425 codelet for RNDIS Vendor information.

---

### Define Documentation

```
#define RNDIS_MAC_ADDRESS
```

RNDIS with MAC address.

Definition at line **81** of file **IxUSBRNDISVendor.h**.

```
#define RNDIS_PRODUCT_ID
```

RNDIS with static Product ID.

Definition at line **67** of file **IxUSBRNDISVendor.h**.

```
#define RNDIS_VENDOR_DESCRIPTION
```

RNDIS with vendor description.

Definition at line **74** of file **IxUSB~~RNDIS~~Vendor.h**.

```
#define RNDIS_VENDOR_ID
```

RNDIS with static Intel Vendor ID.

Definition at line **60** of file **IxUSB~~RNDIS~~Vendor.h**.

# ChannelisedStats Struct Reference

ingroup IxHssAccCodeletCom

## Data Fields

UINT32 **txSamples**  
UINT32 **txBytes**  
UINT32 **rxSamples**  
UINT32 **rxBytes**  
UINT32 **rxIdles**  
UINT32 **rxVerifyFails**  
UINT32 **connectFails**  
UINT32 **portEnableFails**  
UINT32 **portDisableFails**  
UINT32 **disconnectFails**

---

## Detailed Description

ingroup IxHssAccCodeletCom

brief Type definition structure for channelised statistics

Definition at line **92** of file **IxHssAccCodeletCom.h**.

---

The documentation for this struct was generated from the following file:

- **IxHssAccCodeletCom.h**



# GeneralStats Struct Reference

ingroup IxHssAccCodeletCom

## Data Fields

UINT32 **portInitFails**  
UINT32 **errorRetrievalFails**  
UINT32 **txFrmSyncErrors**  
UINT32 **rxFrmSyncErrors**  
UINT32 **txOverRunErrors**  
UINT32 **rxOverRunErrors**  
UINT32 **chanSwTxErrors**  
UINT32 **chanSwRxErrors**  
UINT32 **pktSwTxErrors**  
UINT32 **pktSwRxErrors**  
UINT32 **unrecognisedErrors**

---

## Detailed Description

ingroup IxHssAccCodeletCom

brief Type definition structure for general statistics

Definition at line **68** of file **IxHssAccCodeletCom.h**.

---

The documentation for this struct was generated from the following file:

- **IxHssAccCodeletCom.h**

# ix\_ossl\_thread\_main\_info\_t Struct Reference

## [IXP425 Operating System Services Library (IxOSSL) API]

This type defines thread main info.

### Data Fields

**ix\_ossl\_thread\_entry\_point\_t threadMain**

*pointer to thread entry point function*

**void \* threadArg**

*void pointer to a user defined thread entry point argument*

---

### Detailed Description

This type defines thread main info.

The struct has two fields: threadMain and threadArg. The threadMain is pointer to thread entry point function. threadArg is a void pointer to a user defined thread entry point argument.

Definition at line **262** of file **ix\_ossl.h**.

---

### Field Documentation

**void\* ix\_ossl\_thread\_main\_info\_t::threadArg**

void pointer to a user defined thread entry point argument

Definition at line **265** of file **ix\_ossl.h**.

**ix\_ossl\_thread\_entry\_point\_t ix\_ossl\_thread\_main\_info\_t::threadMain**

pointer to thread entry point function

Definition at line **264** of file **ix\_ossl.h**.

---

The documentation for this struct was generated from the following file:

- **ix\_ossl.h**

# ix\_ossl\_time\_t Struct Reference

## [IXP425 Operating System Services Library (IxOSSL) API]

This type defines OSSL time.

### Data Fields

unsigned long **tv\_sec**  
*seconds*

unsigned long **tv\_nsec**  
*nanoseconds [1, 999,999,999]*

---

### Detailed Description

This type defines OSSL time.

The ix\_ossl\_time\_t struct has two fields. 'sec' and 'nsec'. Time value is computed as:  $\text{sec} * 10^9 + \text{nsec}$

Definition at line **224** of file **ix\_ossl.h**.

---

### Field Documentation

unsigned long ix\_ossl\_time\_t::tv\_nsec

nanoseconds [1, 999,999,999]

Definition at line **227** of file **ix\_ossl.h**.

unsigned long ix\_ossl\_time\_t::tv\_sec

seconds

Definition at line **226** of file **ix\_ossl.h**.

---

The documentation for this struct was generated from the following file:

- **ix\_ossl.h**

# IxAtmCodeletStats Struct Reference

Codelet statistics.

## Data Fields

UINT32 **txPdus**  
UINT32 **txBytes**  
UINT32 **rxPdus**  
UINT32 **rxBytes**  
UINT32 **txDonePdus**  
UINT32 **rxFreeBuffers**  
UINT32 **txPdusSubmitFail**  
UINT32 **txPdusSubmitBusy**  
UINT32 **rxPdusInvalid**

---

## Detailed Description

Codelet statistics.

Definition at line **154** of file **IxAtmCodelet\_p.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmCodelet\_p.h**

# IxAtmdAccUtopiaConfig Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia configuration.

### Data Fields

**IxAtmdAccUtopiaConfig::UtTxConfig\_** **utTxConfig**  
*Tx config Utopia register.*

**IxAtmdAccUtopiaConfig::UtTxStatsConfig\_** **utTxStatsConfig**  
*Tx stats config Utopia register.*

**IxAtmdAccUtopiaConfig::UtTxDefineIdle\_** **utTxDefineIdle**  
*Tx idle cell config Utopia register.*

**IxAtmdAccUtopiaConfig::UtTxEnableFields\_** **utTxEnableFields**  
*Tx enable Utopia register.*

**IxAtmdAccUtopiaConfig::UtTxTransTable0\_** **utTxTransTable0**  
*Tx translation table.*

**IxAtmdAccUtopiaConfig::UtTxTransTable1\_** **utTxTransTable1**  
*Tx translation table.*

**IxAtmdAccUtopiaConfig::UtTxTransTable2\_** **utTxTransTable2**  
*Tx translation table.*

**IxAtmdAccUtopiaConfig::UtTxTransTable3\_** **utTxTransTable3**  
*Tx translation table.*

**IxAtmdAccUtopiaConfig::UtTxTransTable4\_** **utTxTransTable4**  
*Tx translation table.*

**IxAtmdAccUtopiaConfig::UtTxTransTable5\_** **utTxTransTable5**  
*Tx translation table.*

**IxAtmdAccUtopiaConfig::UtRxConfig\_** **utRxConfig**  
*Rx config Utopia register.*

**IxAtmdAccUtopiaConfig::UtRxStatsConfig\_** **utRxStatsConfig**  
*Rx stats config Utopia register.*

**IxAtmdAccUtopiaConfig::UtRxDefineIdle\_** **utRxDefineIdle**  
*Rx idle cell config Utopia register.*

**IxAtmdAccUtopiaConfig::UtRxEnableFields\_** **utRxEnableFields**

*Rx enable Utopia register.*

**IxAtmdAccUtopiaConfig::UtRxTransTable0\_ utRxTransTable0**  
*Rx translation table.*

**IxAtmdAccUtopiaConfig::UtRxTransTable1\_ utRxTransTable1**  
*Rx translation table.*

**IxAtmdAccUtopiaConfig::UtRxTransTable2\_ utRxTransTable2**  
*Rx translation table.*

**IxAtmdAccUtopiaConfig::UtRxTransTable3\_ utRxTransTable3**  
*Rx translation table.*

**IxAtmdAccUtopiaConfig::UtRxTransTable4\_ utRxTransTable4**  
*Rx translation table.*

**IxAtmdAccUtopiaConfig::UtRxTransTable5\_ utRxTransTable5**  
*Rx translation table.*

**IxAtmdAccUtopiaConfig::UtSysConfig\_ utSysConfig**  
*NPE debug config.*

---

## Detailed Description

Utopia configuration.

This structure is used to set the Utopia parameters

- contains the values of Utopia registers, to be set during initialisation
- contains debug commands for NPE, to be used during development steps

*Note:*

- the exact description of all parameters is done in the Utopia reference documents.

Definition at line 937 of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

```
struct IxAtmdAccUtopiaConfig::UtRxConfig_ IxAtmdAccUtopiaConfig::utRxConfig
```

Rx config Utopia register.

```
struct IxAtmdAccUtopiaConfig::UtRxDefineIdle_ IxAtmdAccUtopiaConfig::utRxDefineIdle
```

Rx idle cell config Utopia register.

```
struct IxAtmdAccUtopiaConfig::UtRxEnableFields_ IxAtmdAccUtopiaConfig::utRxEnableFields
```

Rx enable Utopia register.

```
struct IxAtmdAccUtopiaConfig::UtRxStatsConfig_ IxAtmdAccUtopiaConfig::utRxStatsConfig
```

Rx stats config Utopia register.

```
struct IxAtmdAccUtopiaConfig::UtRxTransTable0_ IxAtmdAccUtopiaConfig::utRxTransTable0
```

Rx translation table.

```
struct IxAtmdAccUtopiaConfig::UtRxTransTable1_ IxAtmdAccUtopiaConfig::utRxTransTable1
```

Rx translation table.

```
struct IxAtmdAccUtopiaConfig::UtRxTransTable2_ IxAtmdAccUtopiaConfig::utRxTransTable2
```

Rx translation table.

```
struct IxAtmdAccUtopiaConfig::UtRxTransTable3_ IxAtmdAccUtopiaConfig::utRxTransTable3
```

Rx translation table.

```
struct IxAtmdAccUtopiaConfig::UtRxTransTable4_ IxAtmdAccUtopiaConfig::utRxTransTable4
```

Rx translation table.

```
struct IxAtmdAccUtopiaConfig::UtRxTransTable5_ IxAtmdAccUtopiaConfig::utRxTransTable5
```

Rx translation table.

```
struct IxAtmdAccUtopiaConfig::UtSysConfig_ IxAtmdAccUtopiaConfig::utSysConfig
```

NPE debug config.

```
struct IxAtmdAccUtopiaConfig::UtTxConfig_ IxAtmdAccUtopiaConfig::utTxConfig
```

Tx config Utopia register.

```
struct IxAtmdAccUtopiaConfig::UtTxDefineIdle_ IxAtmdAccUtopiaConfig::utTxDefineIdle
```

Tx idle cell config Utopia register.

```
struct IxAtmdAccUtopiaConfig::UtTxEnableFields_ IxAtmdAccUtopiaConfig::utTxEnableFields
```

Tx enable Utopia register.

```
struct IxAtmdAccUtopiaConfig::UtTxStatsConfig_ IxAtmdAccUtopiaConfig::utTxStatsConfig
```

Tx stats config Utopia register.

```
struct IxAtmdAccUtopiaConfig::UtTxTransTable0_ IxAtmdAccUtopiaConfig::utTxTransTable0
```

Tx translation table.

```
struct IxAtmdAccUtopiaConfig::UtTxTransTable1_ IxAtmdAccUtopiaConfig::utTxTransTable1
```

Tx translation table.

```
struct IxAtmdAccUtopiaConfig::UtTxTransTable2_ IxAtmdAccUtopiaConfig::utTxTransTable2
```

Tx translation table.

```
struct IxAtmdAccUtopiaConfig::UtTxTransTable3_ IxAtmdAccUtopiaConfig::utTxTransTable3
```

Tx translation table.

```
struct IxAtmdAccUtopiaConfig::UtTxTransTable4_ IxAtmdAccUtopiaConfig::utTxTransTable4
```

Tx translation table.

```
struct IxAtmdAccUtopiaConfig::UtTxTransTable5_ IxAtmdAccUtopiaConfig::utTxTransTable5
```



Tx translation table.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtRxConfig\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx config Register.

### Data Fields

unsigned

int **rxInterface**:1

*[31] Utopia Receive Interface*

unsigned

int **rxMode**:1

*[30] Utopia Receive Mode*

unsigned

int **rxOctet**:1

*[29] Utopia Receive cell transfer protocol*

unsigned

int **rxParity**:1

*[28] Utopia Receive Parity Checking enable*

unsigned

int **rxEvenParity**:1

*[27] Utopia Receive Parity Mode*

- 1 – Check for Even Parity
- 0 – Check for Odd Parity.

unsigned

int **rxHEC**:1

*[26] RxHEC Header Error Check Mode*

unsigned

int **rxCOSET**:1

*[25] If enabled the HEC is Exclusive–ORÆed with the value 0x55 before being tested with the received HEC*

unsigned

int **rxHECpass**:1

*[24] Specifies if the incoming cell HEC byte should be transferred after optional processing to the NPE2 Coprocessor Bus Interface or if it should be discarded*

unsigned  
int **reserved\_1**:1  
*[23] These bits are always 0*

unsigned  
int **rxCellSize**:7  
*[22:16] Receive cell size*

unsigned  
int **rxHashEnbGFC**:1  
*[15] Specifies if the VPI field [11:8]/GFC field should be included in the Hash data input or if the bits should be padded with 1Æb0*

unsigned  
int **rxPreHash**:1  
*[14] Enable Pre-hash value generation*

unsigned  
int **reserved\_2**:1  
*[13] These bits are always 0*

unsigned  
int **rxAddrRange**:5  
*[12:8] In ATM master, MPHY mode, this register specifies the upper limit of the PHY polling logical range*

unsigned  
int **reserved\_3**:3  
*[7-5] These bits are always 0 .*

unsigned  
int **rxPHYAddr**:5  
*[4:0] When configured as a slave in an MPHY system this register specifies the physical address of the PHY*

---

## Detailed Description

Utopia Rx config Register.

Definition at line **1274** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig::reserved_1
```

*[23] These bits are always 0*

Definition at line **1317** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::reserved_2
```

[13] These bits are always 0

Definition at line **1334** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::reserved_3
```

[7–5] These bits are always 0 .

Definition at line **1341** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxAddrRange
```

[12:8] In ATM master, MPHY mode, this register specifies the upper limit of the PHY polling logical range

The number of active PHYs are RxAddrRange + 1.

Definition at line **1336** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxCellSize
```

[22:16] Receive cell size

Configures the receive cell size. Values between 52–64 are valid

Definition at line **1319** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxCOSET
```

[25] If enabled the HEC is Exclusive–ORÆed with the value 0x55 before being tested with the received HEC

- 1 – Enable HEC ExOR with value 0x55.
- 0 – Use generated HEC value.

Definition at line **1306** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxEvenParity
```

[27] Utopia Receive Parity Mode

- 1 – Check for Even Parity
- 0 – Check for Odd Parity.

Definition at line **1297** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxHashEnbGFC
```

[15] Specifies if the VPI field [11:8]/GFC field should be included in the Hash data input or if the bits should be padded with 1Æb0

- 1 – VPI [11:8]/GFC field valid and used in Hash residue calculation.
- 0 – VPI [11:8]/GFC field padded with 1Æb0

Definition at line **1322** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxHEC
```

[26] RxHEC Header Error Check Mode

Enables/disables cell header error checking on the received cell header.

- 1 – HEC checking enabled
- 0 – HEC checking disabled

Definition at line **1301** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxHECpass
```

[24] Specifies if the incoming cell HEC byte should be transferred after optional processing to the NPE2 Coprocessor Bus Interface or if it should be discarded

- 1 – HEC maintained 53-byte/UDC cell sent to NPE2.
- 0 – HEC discarded 52-byte/UDC cell sent to NPE2 coprocessor.

Definition at line **1311** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxInterface
```

[31] Utopia Receive Interface

The following encoding is used to set the Utopia Receive interface as ATM master or PHY slave:

- 1 – PHY
- 0 – ATM

Definition at line **1277** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxMode
```

[30] Utopia Receive Mode

The following encoding is used to set the Utopia Receive mode to SPHY or MPHY:

- 1 – SPHY
- 0 – MPHY

Definition at line **1282** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxOctet
```

[29] Utopia Receive cell transfer protocol

Used to set the Utopia cell transfer protocol to Octet–level handshaking. Note this is only applicable in SPHY mode.

- 1 – Octet–handshaking enabled
- 0 – Cell–handshaking enabled

Definition at line **1287** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxParity
```

[28] Utopia Receive Parity Checking enable

- 1 – Parity checking enabled
- 0 – Parity checking disabled

Definition at line **1293** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxPHYAddr
```

[4:0] When configured as a slave in an MPHY system this register specifies the physical address of the PHY

Definition at line **1342** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxConfig_::rxPreHash
```

[14] Enable Pre–hash value generation

Specifies if the incoming cell data should be pre-hashed to allow VPI/VCI header look-up in a hash table.

- 1 – Pre-hashing enabled
- 0 – Pre-hashing disabled

Definition at line **1328** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtRxDefinIdle\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx idle cells config Register.

### Data Fields

unsigned int **vpi**:12

*[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]*

- *Note: if VCIdleRxGFC is set to 0 the GFC field is ignored in test*

unsigned int **vci**:16

*[19:4] ATM VCI [15:0]*

unsigned int **pti**:3

*[3:1] ATM PTI PTI [2:0]*

- *Note: if VCIdleRxPTI is set to 0 the PTI field is ignored in test.*

unsigned int **clp**:1

*[0] ATM CLP [0]*

- *Note: if VCIdleRxCLP is set to 0 the CLP field is ignored in test.*

---

## Detailed Description

Utopia Rx idle cells config Register.

Definition at line **1376** of file **IxAtmdAccCtrl.h**.

---



## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxDefineIdle\_::clp

[0] ATM CLP [0]

- Note: if VCIdleRxCLP is set to 0 the CLP field is ignored in test.

Definition at line **1387** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxDefineIdle\_::pti

[3:1] ATM PTI PTI [2:0]

- Note: if VCIdleRxPTI is set to 0 the PTI field is ignored in test.

Definition at line **1384** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxDefineIdle\_::vci

[19:4] ATM VCI [15:0]

Definition at line **1382** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxDefineIdle\_::vpi

[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]

- Note: if VCIdleRxGFC is set to 0 the GFC field is ignored in test

Definition at line **1379** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtRxEnableFields\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx enable Register.

### Data Fields

unsigned

int **defineRxIdleGFC**:1

*[31] This register is used to include or exclude the GFC field of the ATM header when testing for Idle cells*

unsigned

int **defineRxIdlePTI**:1

*[30] This register is used to include or exclude the PTI field of the ATM header when testing for Idle cells*

unsigned

int **defineRxIdleCLP**:1

*[29] This register is used to include or exclude the CLP field of the ATM header when testing for Idle cells*

unsigned

int **phyStatsRxEnb**:1

*[28] This register is used to enable or disable ATM statistics gathering based on the specified PHY address as defined in RxStatsConfig register*

unsigned

int **vcStatsRxEnb**:1

*[27] This register is used to enable or disable ATM statistics gathering based on a specific VPI/VCI address*

unsigned

int **vcStatsRxGFC**:1

*[26] This register is used to include or exclude the GFC field of the ATM header when ATM VPI/VCI statistics are enabled*

unsigned

int **vcStatsRxPTI**:1

*[25] This register is used to include or exclude the PTI field of the ATM header when ATM VPI/VCI statistics are enabled*

unsigned

int **vcStatsRxCLP**:1

*[24] This register is used to include or exclude the CLP field of the ATM header when ATM VPI/VCI statistics are enabled*

unsigned

int **discardHecErr**:1

*[23] Discard cells with an invalid HEC*

unsigned

int **discardParErr**:1

*[22] Discard cells containing parity errors*

unsigned

int **discardIdle**:1

*[21] Discard Idle Cells based on DefineIdle register values*

- 1 – Discard IDLE cells
- 0 – IDLE cells passed

unsigned

int **enbHecErrCnt**:1

*[20] Enable Receive HEC Error Count*

unsigned

int **enbParErrCnt**:1

*[19] Enable Parity Error Count*

- 1 – Enable count of received cells containing Parity errors
- 0 – No count is maintained

unsigned

int **enbIdleCellCnt**:1

*[18] Enable Receive Idle Cell Count*

unsigned

int **enbSizeErrCnt**:1

*[17] Enable Receive Size Error Count*

unsigned

int **enbRxCellCnt**:1

*[16] Enable Receive Valid Cell Count of non-idle/non-error cells*

unsigned

int **reserved\_1**:3

*[15:13] These bits are always 0*

unsigned

int **rxCellOvrInt**:1

*[12] Enable CBI Utopia Receive Status Condition if the RxCellCount register overflows*

unsigned  
int **invalidHecOvrInt**:1  
*[11] Enable CBI Receive Status Condition if the InvalidHecCount register overflows*

unsigned  
int **invalidParOvrInt**:1  
*[10] Enable CBI Receive Status Condition if the InvalidParCount register overflows*

- 1 – CBI Receive Condition asserted

unsigned  
int **invalidSizeOvrInt**:1  
*[9] Enable CBI Receive Status Condition if the InvalidSizeCount register overflows*

unsigned  
int **rxIdleOvrInt**:1  
*[8] Enable CBI Receive Status Condition if the RxIdleCount overflows*

unsigned  
int **reserved\_2**:3  
*[7:5] These bits are always 0*

unsigned  
int **rxAddrMask**:5  
*[4:0] This register is used as a mask to allow the user to increase the PHY receive address range*

---

## Detailed Description

Utopia Rx enable Register.

Definition at line **1396** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtrxEnableFields\_::defineRxIdleCLP

[29] This register is used to include or exclude the CLP field of the ATM header when testing for Idle cells

- 1 – CLP field is valid.
- 0 – CLP field ignored.

Definition at line **1409** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::defineRxIdleGFC
```

[31] This register is used to include or exclude the GFC field of the ATM header when testing for Idle cells

- 1 – GFC field is valid.
- 0 – GFC field ignored.

Definition at line **1399** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::defineRxIdlePTI
```

[30] This register is used to include or exclude the PTI field of the ATM header when testing for Idle cells

- 1 – PTI field is valid.
- 0 – PTI field ignored.

Definition at line **1404** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::discardHecErr
```

[23] Discard cells with an invalid HEC

- 1 – Discard cells with HEC errors
- 0 – Cells with HEC errors are passed

Definition at line **1441** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::discardIdle
```

[21] Discard Idle Cells based on DefineIdle register values

- 1 – Discard IDLE cells
- 0 – IDLE cells passed

Definition at line **1449** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::discardParErr
```

[22] Discard cells containing parity errors

- 1 – Discard cells with parity errors
- 0 – Cells with parity errors are passed

Definition at line **1445** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields\_::enbHecErrCnt

[20] Enable Receive HEC Error Count

- 1 – Enable count of received cells containing HEC errors
- 0 – No count is maintained.

Definition at line **1453** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields\_::enbIdleCellCnt

[18] Enable Receive Idle Cell Count

- 1 – Enable count of Idle cells received.
- 0 – No count is maintained.

Definition at line **1461** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields\_::enbParErrCnt

[19] Enable Parity Error Count

- 1 – Enable count of received cells containing Parity errors
- 0 – No count is maintained

Definition at line **1457** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields\_::enbRxCellCnt

[16] Enable Receive Valid Cell Count of non-idle/non-error cells

- 1 – Enable count of valid cells received – non-idle/non-error
- 0 – No count is maintained.

Definition at line **1469** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields\_::enbSizeErrCnt

[17] Enable Receive Size Error Count

- 1 – Enable count of received cells of incorrect size

- 0 – No count is maintained.

Definition at line **1465** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtrxEnableFields_::invalidHecOvrInt
```

[11] Enable CBI Receive Status Condition if the InvalidHecCount register overflows

- 1 – CBI Receive Condition asserted.
- 0 – No CBI Receive Condition asserted

Definition at line **1480** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtrxEnableFields_::invalidParOvrInt
```

[10] Enable CBI Receive Status Condition if the InvalidParCount register overflows

- 1 – CBI Receive Condition asserted
- 0 – No CBI Receive Condition asserted

Definition at line **1485** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtrxEnableFields_::invalidSizeOvrInt
```

[9] Enable CBI Receive Status Condition if the InvalidSizeCount register overflows

- 1 – CBI Receive Status Condition asserted.
- 0 – No CBI Receive Status asserted

Definition at line **1490** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtrxEnableFields_::phyStatsRxEnb
```

[28] This register is used to enable or disable ATM statistics gathering based on the specified PHY address as defined in RxStatsConfig register

- 1 – Enable statistics for specified receive PHY address.
- 0 – Disable statistics for specified receive PHY address.

Definition at line **1414** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtrxEnableFields_::reserved_1
```

[15:13] These bits are always 0

Definition at line **1473** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::reserved_2
```

[7:5] These bits are always 0

Definition at line **1499** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::rxAddrMask
```

[4:0] This register is used as a mask to allow the user to increase the PHY receive address range

The register should be programmed with the address-range limit, i.e. if set to 0x3 the address range increases to a maximum of 4 addresses.

Definition at line **1501** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::rxCellOvrInt
```

[12] Enable CBI Utopia Receive Status Condition if the RxCellCount register overflows

- 1 – CBI Receive Status asserted.
- 0 – No CBI Receive Status asserted.

Definition at line **1475** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::rxIdleOvrInt
```

[8] Enable CBI Receive Status Condition if the RxIdleCount overflows

- 1 – CBI Receive Condition asserted.
- 0 – No CBI Receive Condition asserted

Definition at line **1495** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::vcStatsRxCLP
```

[24] This register is used to include or exclude the CLP field of the ATM header when ATM VPI/VCI statistics are enabled

- 1 – CLP field is valid.
- 0 – CLP field ignored.



Definition at line **1436** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::vcStatsRxEnb
```

[27] This register is used to enable or disable ATM statistics gathering based on a specific VPI/VCI address

- 1 – Enable statistics for specified VPI/VCI address.
- 0 – Disable statistics for specified VPI/VCI address.

Definition at line **1420** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::vcStatsRxGFC
```

[26] This register is used to include or exclude the GFC field of the ATM header when ATM VPI/VCI statistics are enabled

GFC is only available at the UNI and uses the first 4-bits of the VPI field.

- 1 – GFC field is valid.
- 0 – GFC field ignored.

Definition at line **1425** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxEnableFields_::vcStatsRxPTI
```

[25] This register is used to include or exclude the PTI field of the ATM header when ATM VPI/VCI statistics are enabled

- 1 – PTI field is valid.
- 0 – PTI field ignored.

Definition at line **1431** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtRxStatsConfig\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx stats config Register.

### Data Fields

unsigned int **vpi**:12

*[31:20] ATM VPI VPI [11:0] OR GFC [3:0] and VPI [7:0]*

- *Note: if VCStatsRxGFC is set to 0 the GFC field is ignored in test*

unsigned int **vci**:16

*[19:4] VCI [15:0] or PHY Address [4]*

unsigned int **pti**:3

*[3:1] PTI [2:0] or or PHY Address [3:1]*

- *Note: if VCStatsRxPTI is set to 0 the PTI field is ignored in test*

unsigned int **clp**:1

*[0] CLP [0] or PHY Address [0]*

- *Note: if VCStatsRxCLP is set to 0 the CLP field is ignored in test*

---

## Detailed Description

Utopia Rx stats config Register.

Definition at line **1352** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxStatsConfig\_::clp

[0] CLP [0] or PHY Address [0]

- Note: if VCStatsRxCLP is set to 0 the CLP field is ignored in test
- Note: if VCStatsRxEnb is set to 0 only the PHY port address is used for statistics gathering..

Definition at line **1365** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxStatsConfig\_::pti

[3:1] PTI [2:0] or or PHY Address [3:1]

- Note: if VCStatsRxPTI is set to 0 the PTI field is ignored in test
- Note: if VCStatsRxEnb is set to 0 only the PHY port address is used for statistics gathering..

Definition at line **1360** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxStatsConfig\_::vci

[19:4] VCI [15:0] or PHY Address [4]

Definition at line **1358** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxStatsConfig\_::vpi

[31:20] ATM VPI VPI [11:0] OR GFC [3:0] and VPI [7:0]

- Note: if VCStatsRxGFC is set to 0 the GFC field is ignored in test

Definition at line **1355** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtRxTransTable0\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

### Data Fields

- unsigned int **phy0**:5  
[31–27] Rx Mapping value of logical phy 0
- unsigned int **phy1**:5  
[26–22] Rx Mapping value of logical phy 1
- unsigned int **phy2**:5  
[21–17] Rx Mapping value of logical phy 2
- unsigned int **reserved\_1**:1  
[16] These bits are always 0
- unsigned int **phy3**:5  
[15–11] Rx Mapping value of logical phy 3
- unsigned int **phy4**:5  
[10–6] Rx Mapping value of logical phy 4
- unsigned int **phy5**:5  
[5–1] Rx Mapping value of logical phy 5
- unsigned int **reserved\_2**:1  
[0] These bits are always 0

---

### Detailed Description

Utopia Rx translation table Register.

Definition at line 1512 of file **IxAtmdAccCtrl.h**.

---

### Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0\_::phy0

[31–27] Rx Mapping value of logical phy 0

Definition at line **1515** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy1
```

[26–22] Rx Mapping value of logical phy 1

Definition at line **1517** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy2
```

[21–17] Rx Mapping value of logical phy 2

Definition at line **1519** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy3
```

[15–11] Rx Mapping value of logical phy 3

Definition at line **1523** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy4
```

[10–6] Rx Mapping value of logical phy 4

Definition at line **1525** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::phy5
```

[5–1] Rx Mapping value of logical phy 5

Definition at line **1527** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::reserved_1
```

[16] These bits are always 0

Definition at line **1521** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable0_::reserved_2
```

[0] These bits are always 0

Definition at line **1529** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtRxTransTable1\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

### Data Fields

- unsigned int **phy6**:5  
*[31–27] Rx Mapping value of logical phy 6*
- unsigned int **phy7**:5  
*[26–22] Rx Mapping value of logical phy 7*
- unsigned int **phy8**:5  
*[21–17] Rx Mapping value of logical phy 8*
- unsigned int **reserved\_1**:1  
*[16–0] These bits are always 0*
- unsigned int **phy9**:5  
*[15–11] Rx Mapping value of logical phy 3*
- unsigned int **phy10**:5  
*[10–6] Rx Mapping value of logical phy 4*
- unsigned int **phy11**:5  
*[5–1] Rx Mapping value of logical phy 5*
- unsigned int **reserved\_2**:1  
*[0] These bits are always 0*

---

### Detailed Description

Utopia Rx translation table Register.

Definition at line **1539** of file **IxAtmdAccCtrl.h**.

---

### Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1\_::phy10

[10–6] Rx Mapping value of logical phy 4

Definition at line **1552** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy11
```

[5–1] Rx Mapping value of logical phy 5

Definition at line **1554** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy6
```

[31–27] Rx Mapping value of logical phy 6

Definition at line **1542** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy7
```

[26–22] Rx Mapping value of logical phy 7

Definition at line **1544** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy8
```

[21–17] Rx Mapping value of logical phy 8

Definition at line **1546** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::phy9
```

[15–11] Rx Mapping value of logical phy 3

Definition at line **1550** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::reserved_1
```

[16–0] These bits are always 0

Definition at line **1548** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable1_::reserved_2
```



[0] These bits are always 0

Definition at line **1556** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtRxTransTable2\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

### Data Fields

- unsigned int **phy12**:5  
*[31–27] Rx Mapping value of logical phy 6*
- unsigned int **phy13**:5  
*[26–22] Rx Mapping value of logical phy 7*
- unsigned int **phy14**:5  
*[21–17] Rx Mapping value of logical phy 8*
- unsigned int **reserved\_1**:1  
*[16–0] These bits are always 0*
- unsigned int **phy15**:5  
*[15–11] Rx Mapping value of logical phy 3*
- unsigned int **phy16**:5  
*[10–6] Rx Mapping value of logical phy 4*
- unsigned int **phy17**:5  
*[5–1] Rx Mapping value of logical phy 5*
- unsigned int **reserved\_2**:1  
*[0] These bits are always 0*

---

### Detailed Description

Utopia Rx translation table Register.

Definition at line **1566** of file **IxAtmdAccCtrl.h**.

---

### Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable2\_::phy12

[31–27] Rx Mapping value of logical phy 6

Definition at line **1569** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable2\_::phy13

[26–22] Rx Mapping value of logical phy 7

Definition at line **1571** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable2\_::phy14

[21–17] Rx Mapping value of logical phy 8

Definition at line **1573** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable2\_::phy15

[15–11] Rx Mapping value of logical phy 3

Definition at line **1577** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable2\_::phy16

[10–6] Rx Mapping value of logical phy 4

Definition at line **1579** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable2\_::phy17

[5–1] Rx Mapping value of logical phy 5

Definition at line **1581** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable2\_::reserved\_1

[16–0] These bits are always 0

Definition at line **1575** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable2\_::reserved\_2

[0] These bits are always 0

Definition at line **1583** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtRxTransTable3\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

### Data Fields

- unsigned int **phy18**:5  
*[31–27] Rx Mapping value of logical phy 6*
- unsigned int **phy19**:5  
*[26–22] Rx Mapping value of logical phy 7*
- unsigned int **phy20**:5  
*[21–17] Rx Mapping value of logical phy 8*
- unsigned int **reserved\_1**:1  
*[16–0] These bits are always 0*
- unsigned int **phy21**:5  
*[15–11] Rx Mapping value of logical phy 3*
- unsigned int **phy22**:5  
*[10–6] Rx Mapping value of logical phy 4*
- unsigned int **phy23**:5  
*[5–1] Rx Mapping value of logical phy 5*
- unsigned int **reserved\_2**:1  
*[0] These bits are always 0*

---

### Detailed Description

Utopia Rx translation table Register.

Definition at line **1591** of file **IxAtmdAccCtrl.h**.

---

### Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable3\_::phy18

[31–27] Rx Mapping value of logical phy 6

Definition at line **1594** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable3\_::phy19

[26–22] Rx Mapping value of logical phy 7

Definition at line **1596** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable3\_::phy20

[21–17] Rx Mapping value of logical phy 8

Definition at line **1598** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable3\_::phy21

[15–11] Rx Mapping value of logical phy 3

Definition at line **1602** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable3\_::phy22

[10–6] Rx Mapping value of logical phy 4

Definition at line **1604** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable3\_::phy23

[5–1] Rx Mapping value of logical phy 5

Definition at line **1606** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable3\_::reserved\_1

[16–0] These bits are always 0

Definition at line **1600** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable3\_::reserved\_2

[0] These bits are always 0

Definition at line **1608** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtRxTransTable4\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

### Data Fields

- unsigned int **phy24**:5  
*[31–27] Rx Mapping value of logical phy 6*
- unsigned int **phy25**:5  
*[26–22] Rx Mapping value of logical phy 7*
- unsigned int **phy26**:5  
*[21–17] Rx Mapping value of logical phy 8*
- unsigned int **reserved\_1**:1  
*[16–0] These bits are always 0*
- unsigned int **phy27**:5  
*[15–11] Rx Mapping value of logical phy 3*
- unsigned int **phy28**:5  
*[10–6] Rx Mapping value of logical phy 4*
- unsigned int **phy29**:5  
*[5–1] Rx Mapping value of logical phy 5*
- unsigned int **reserved\_2**:1  
*[0] These bits are always 0*

---

### Detailed Description

Utopia Rx translation table Register.

Definition at line **1616** of file **IxAtmdAccCtrl.h**.

---

### Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable4\_::phy24



[31–27] Rx Mapping value of logical phy 6

Definition at line **1619** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable4\_::phy25

[26–22] Rx Mapping value of logical phy 7

Definition at line **1621** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable4\_::phy26

[21–17] Rx Mapping value of logical phy 8

Definition at line **1623** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable4\_::phy27

[15–11] Rx Mapping value of logical phy 3

Definition at line **1627** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable4\_::phy28

[10–6] Rx Mapping value of logical phy 4

Definition at line **1629** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable4\_::phy29

[5–1] Rx Mapping value of logical phy 5

Definition at line **1631** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable4\_::reserved\_1

[16–0] These bits are always 0

Definition at line **1625** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtrxTransTable4\_::reserved\_2

[0] These bits are always 0

Definition at line **1633** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtRxTransTable5\_ Struct Reference

[IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx translation table Register.

## Data Fields

unsigned int **phy30**:5  
*[31–27] Rx Mapping value of logical phy 6*

unsigned int **reserved\_1**:27  
*[26–0] These bits are always 0*

---

## Detailed Description

Utopia Rx translation table Register.

Definition at line **1641** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable5\_::phy30

[31–27] Rx Mapping value of logical phy 6

Definition at line **1644** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtRxTransTable5\_::reserved\_1

[26–0] These bits are always 0

Definition at line **1646** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtSysConfig\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

NPE setup Register.

### Data Fields

unsigned int **reserved\_1**:2

*[31–30] These bits are always 0*

unsigned int **txEnbFSM**:1

*[29] Enables the operation of the Utopia Transmit FSM*

- 1 – FSM enabled
- 0 – FSM inactive

unsigned int **rxEnbFSM**:1

*[28] Enables the operation of the Utopia Receive FSM*

- 1 – FSM enabled
- 0 – FSM inactive

unsigned int **disablePins**:1

*[27] Disable Utopia interface I/O pins forcing the signals to an inactive state*

unsigned int **tstLoop**:1

*[26] Test Loop Back Enable*

unsigned int **txReset**:1

*[25] Resets the Utopia Coprocessor transmit module to a known state*

unsigned int **rxReset**:1

*[24] Resets the Utopia Coprocessor receive module to a known state*

unsigned int **reserved\_2**:24

*[23–0] These bits are always 0*

## Detailed Description

NPE setup Register.

Definition at line **1655** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig\_::disablePins

[27] Disable Utopia interface I/O pins forcing the signals to an inactive state

Note that this bit is set on reset and must be de-asserted

- 0 – Normal data transfer
- 1 – Utopia interface pins are forced inactive

Definition at line **1667** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig\_::reserved\_1

[31–30] These bits are always 0

Definition at line **1658** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig\_::reserved\_2

[23–0] These bits are always 0

Definition at line **1694** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig\_::rxEnbFSM

[28] Enables the operation of the Utopia Receive FSM

- 1 – FSM enabled
- 0 – FSM inactive

Definition at line **1663** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtSysConfig\_::rxReset

[24] Resets the Utopia Coprocessor receive module to a known state

- Note: All receive configuration and status registers will be reset to their reset values.
- 0 – Normal operating mode
- 1 – Reset receive modules

Definition at line **1687** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::tstLoop
```

[26] Test Loop Back Enable

- Note: For loop back to function RxMode and Tx Mode must both be set to single PHY mode.
- 0 – Loop back
- 1 – Normal operating mode

Definition at line **1673** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::txEnbFSM
```

[29] Enables the operation of the Utopia Transmit FSM

- 1 – FSM enabled
- 0 – FSM inactive

Definition at line **1659** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtSysConfig_::txReset
```

[25] Resets the Utopia Coprocessor transmit module to a known state

- Note: All transmit configuration and status registers will be reset to their reset values.
- 0 – Normal operating mode,
- 1 – Reset transmit modules

Definition at line **1680** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtTxConfig\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx Config Register.

### Data Fields

unsigned

int **reserved\_1**:1

*[31] These bits are always 0.*

unsigned

int **txInterface**:1

*[30] Utopia Transmit Interface*

unsigned

int **txMode**:1

*[29] Utopia Transmit Mode*

unsigned

int **txOctet**:1

*[28] Utopia Transmit cell transfer protocol*

unsigned

int **txParity**:1

*[27] Utopia Transmit parity enabled when set*

unsigned

int **txEvenParity**:1

*[26] Utopia Transmit Parity Mode*

- 1 – Even Parity Generated

unsigned

int **txHEC**:1

*[25] Header Error Check Insertion Mode*

unsigned

int **txCOSET**:1

*[24] If enabled the HEC is Exclusive-ORÆed with the value 0x55 before being presented on the Utopia bus*

unsigned

int **reserved\_2**:1

*[23] These bits are always 0*

unsigned

int **txCellSize**:7

*[22:16] Transmit expected cell size*

unsigned

int **reserved\_3**:3

*[15:13] These bits are always 0*

unsigned

int **txAddrRange**:5

*[12:8] When configured as an ATM master in MPHY mode this register specifies the upper limit of the PHY polling logical range*

unsigned

int **reserved\_4**:3

*[7:5] These bits are always 0*

unsigned

int **txPHYAddr**:5

*[4:0] When configured as a slave in an MPHY system this register specifies the physical address of the PHY*

---

## Detailed Description

Utopia Tx Config Register.

Definition at line **944** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig::reserved\_1

*[31] These bits are always 0.*

Definition at line **947** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig::reserved\_2

*[23] These bits are always 0*

Definition at line **985** of file **IxAtmdAccCtrl.h**.



unsigned int IxAtmdAccUtopiaConfig::UtTxConfig\_::reserved\_3

[15:13] These bits are always 0

Definition at line **990** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig\_::reserved\_4

[7:5] These bits are always 0

Definition at line **995** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig\_::txAddrRange

[12:8] When configured as an ATM master in MPHY mode this register specifies the upper limit of the PHY polling logical range

The number of active PHYs are TxAddrRange + 1.

Definition at line **991** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig\_::txCellSize

[22:16] Transmit expected cell size

Configures the cell size for the transmit module: Values between 52–64 are valid.

Definition at line **987** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig\_::txCOSET

[24] If enabled the HEC is Exclusive-OR'ed with the value 0x55 before being presented on the Utopia bus

- 1 – Enable HEC ExOR with value 0x55
- 0 – Use generated HEC value.

Definition at line **979** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxConfig\_::txEvenParity

[26] Utopia Transmit Parity Mode

- 1 – Even Parity Generated

- 0 – Odd Parity Generated.

Definition at line **970** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txHEC
```

[25] Header Error Check Insertion Mode

Specifies if the transmit cell header check byte is calculated and inserted when set.

- 1 – Generate HEC.
- 0 – Disable HEC generation.

Definition at line **974** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txInterface
```

[30] Utopia Transmit Interface

The following encoding is used to set the Utopia Transmit interface as ATM master or PHY slave:

- 1 – PHY
- 0 – ATM

Definition at line **948** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txMode
```

[29] Utopia Transmit Mode

The following encoding is used to set the Utopia Transmit mode to SPHY or MPHY:

- 1 – SPHY
- 0 – MPHY

Definition at line **954** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txOctet
```

[28] Utopia Transmit cell transfer protocol

Used to set the Utopia cell transfer protocol to Octet-level handshaking. Note this is only applicable in SPHY mode.

- 1 – Octet-handshaking enabled

- 0 – Cell–handshaking enabled

Definition at line **959** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txParity
```

[27] Utopia Transmit parity enabled when set

TxEvenParity defines the parity format odd/even.

- 1 – Enable Parity generation.
- 0 – ut\_op\_prtly held low.

Definition at line **965** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxConfig_::txPHYAddr
```

[4:0] When configured as a slave in an MPHY system this register specifies the physical address of the PHY

Definition at line **996** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtTxDefinIdle\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx idle cells Register.

### Data Fields

unsigned int **vpi**:12

*[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]*

- *Note: if VCIdleTxGFC is set to 0 the GFC field is ignored in test*

unsigned int **vci**:16

*[19:4] ATM VCI [15:0]*

unsigned int **pti**:3

*[3:1] ATM PTI PTI [2:0]*

- *Note: if VCIdleTxPTI is set to 0 the PTI field is ignored in test.*

unsigned int **clp**:1

*[0] ATM CLP [0]*

- *Note: if VCIdleTxCLP is set to 0 the CLP field is ignored in test.*

---

## Detailed Description

Utopia Tx idle cells Register.

Definition at line **1034** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxDefineIdle\_::clp

[0] ATM CLP [0]

- Note: if VCIdleTxCLP is set to 0 the CLP field is ignored in test.

Definition at line **1045** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxDefineIdle\_::pti

[3:1] ATM PTI PTI [2:0]

- Note: if VCIdleTxPTI is set to 0 the PTI field is ignored in test.

Definition at line **1042** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxDefineIdle\_::vci

[19:4] ATM VCI [15:0]

Definition at line **1040** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxDefineIdle\_::vpi

[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]

- Note: if VCIdleTxGFC is set to 0 the GFC field is ignored in test

Definition at line **1037** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtTxEnableFields\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx ienable fields Register.

### Data Fields

unsigned int **defineTxIdleGFC**:1

*[31] This register is used to include or exclude the GFC field of the ATM header when testing for Idle cells*

unsigned int **defineTxIdlePTI**:1

*[30] This register is used to include or exclude the PTI field of the ATM header when testing for Idle cells*

unsigned int **defineTxIdleCLP**:1

*[29] This register is used to include or exclude the CLP field of the ATM header when testing for Idle cells*

unsigned int **phyStatsTxEnb**:1

*[28] This register is used to enable or disable ATM statistics gathering based on the specified PHY address as defined in TxStatsConfig register*

unsigned int **vcStatsTxEnb**:1

*[27] This register is used to change the ATM statistics–gathering mode from the specified logical PHY address to a specific VPI/VCI address*

unsigned int **vcStatsTxGFC**:1

*[26] This register is used to include or exclude the GFC field of the ATM header when ATM VPI/VCI statistics are enabled*

unsigned int **vcStatsTxPTI**:1

*[25] This register is used to include or exclude the PTI field of the ATM header when ATM VPI/VCI statistics are enabled*

unsigned int **vcStatsTxCLP**:1

*[24] This register is used to include or exclude the CLP field of the ATM header when ATM VPI/VCI statistics are enabled*

unsigned int **reserved\_1**:3

*[23–21] These bits are always 0*

unsigned int **txPollStsInt**:1

*[20] Enable the assertion of the ucp\_tx\_poll\_sts condition where there is a change in polling status*

unsigned int **txCellOvrInt**:1

*[19] Enable TxCellCount overflow CBI Transmit Status condition assertion*

unsigned int **txIdleCellOvrInt**:1

*[18] Enable TxIdleCellCount overflow Transmit Status Condition*

- 1 – If TxIdleCellCountOvr is set assert the Transmit Status Condition
- 0 – No CBI Transmit Status condition assertion..

unsigned int **enbIdleCellCnt**:1

*[17] Enable Transmit Idle Cell Count*

unsigned int **enbTxCellCnt**:1

*[16] Enable Transmit Valid Cell Count of non-idle/non-error cells*

- 1 – Enable count of valid cells transmitted– non-idle/non-error
- 0 – No count is maintained.

unsigned int **reserved\_2**:16

*[15:0] These bits are always 0*

---

## Detailed Description

Utopia Tx ienable fields Register.

Definition at line **1056** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields\_::defineTxIdleCLP

[29] This register is used to include or exclude the CLP field of the ATM header when testing for Idle cells

- 1 – CLP field is valid.
- 0 – CLP field ignored.

Definition at line **1069** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields\_::defineTxIdleGFC

[31] This register is used to include or exclude the GFC field of the ATM header when testing for Idle cells

- 1 – GFC field is valid.
- 0 – GFC field ignored.

Definition at line **1059** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::defineTxIdlePTI
```

[30] This register is used to include or exclude the PTI field of the ATM header when testing for Idle cells

- 1 – PTI field is valid
- 0 – PTI field ignored.

Definition at line **1064** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::enbIdleCellCnt
```

[17] Enable Transmit Idle Cell Count

- 1 – Enable count of Idle cells transmitted.
- 0 – No count is maintained.

Definition at line **1119** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::enbTxCellCnt
```

[16] Enable Transmit Valid Cell Count of non-idle/non-error cells

- 1 – Enable count of valid cells transmitted–non-idle/non-error
- 0 – No count is maintained.

Definition at line **1123** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::phyStatsTxEnb
```

[28] This register is used to enable or disable ATM statistics gathering based on the specified PHY address as defined in TxStatsConfig register

- 1 – Enable statistics for specified transmit PHY address.
- 0 – Disable statistics for specified transmit PHY address.

Definition at line **1074** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::reserved_1
```



[23–21] These bits are always 0

Definition at line **1103** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::reserved_2
```

[15:0] These bits are always 0

Definition at line **1127** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::txCellOvrInt
```

[19] Enable TxCellCount overflow CBI Transmit Status condition assertion

- 1 – If TxCellCountOvr is set assert the Transmit Status Condition.
- 0 – No CBI Transmit Status condition assertion

Definition at line **1110** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::txIdleCellOvrInt
```

[18] Enable TxIdleCellCount overflow Transmit Status Condition

- 1 – If TxIdleCellCountOvr is set assert the Transmit Status Condition
- 0 – No CBI Transmit Status condition assertion..

Definition at line **1115** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::txPollStsInt
```

[20] Enable the assertion of the ucp\_tx\_poll\_sts condition where there is a change in polling status

- 1 – ucp\_tx\_poll\_sts asserted whenever there is a change in status
- 0 – ucp\_tx\_poll\_sts asserted if ANY transmit PHY is available

Definition at line **1105** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::vcStatsTxCLP
```

[24] This register is used to include or exclude the CLP field of the ATM header when ATM VPI/VCI statistics are enabled

- 1 – CLP field is valid

- 0 – CLP field ignored.

Definition at line **1098** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::vcStatsTxEnb
```

[27] This register is used to change the ATM statistics–gathering mode from the specified logical PHY address to a specific VPI/VCI address

- 1 – Enable statistics for specified VPI/VCI address.
- 0 – Disable statistics for specified VPI/VCI address

Definition at line **1080** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::vcStatsTxGFC
```

[26] This register is used to include or exclude the GFC field of the ATM header when ATM VPI/VCI statistics are enabled

GFC is only available at the UNI and uses the first 4–bits of the VPI field.

- 1 – GFC field is valid
- 0 – GFC field ignored.

Definition at line **1086** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxEnableFields_::vcStatsTxPTI
```

[25] This register is used to include or exclude the PTI field of the ATM header when ATM VPI/VCI statistics are enabled

- 1 – PTI field is valid
- 0 – PTI field ignored.

Definition at line **1093** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtTxStatsConfig\_ Struct Reference

[IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx stats Register.

## Data Fields

unsigned int **vpi**:12

*[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]*

- *Note: if VCStatsTxGFC is set to 0 the GFC field is ignored in test*

unsigned int **vci**:16

*[19:4] ATM VCI [15:0] or PHY Address[4]*

unsigned int **pti**:3

*[3:1] ATM PTI [2:0] or PHY Address[3:1]*

- *Note: if VCStatsTxPTI is set to 0 the PTI field is ignored in test*

unsigned int **clp**:1

*[0] ATM CLP or PHY Address [0]*

- *Note: if VCStatsTxCLP is set to 0 the CLP field is ignored in test*

---

## Detailed Description

Utopia Tx stats Register.

Definition at line **1008** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxStatsConfig\_::clp

[0] ATM CLP or PHY Address [0]

- Note: if VCStatsTxCLP is set to 0 the CLP field is ignored in test
- Note: if VCStatsTxEnb is set to 0 only the transmit PHY port address as defined by this register is used for ATM statistics [4:0].

Definition at line **1021** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxStatsConfig\_::pti

[3:1] ATM PTI [2:0] or PHY Address[3:1]

- Note: if VCStatsTxPTI is set to 0 the PTI field is ignored in test
- Note: if VCStatsTxEnb is set to 0 only the transmit PHY port address as defined by this register is used for ATM statistics [4:0].

Definition at line **1016** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxStatsConfig\_::vci

[19:4] ATM VCI [15:0] or PHY Address[4]

Definition at line **1014** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxStatsConfig\_::vpi

[31:20] ATM VPI [11:0] OR GFC [3:0] and VPI [7:0]

- Note: if VCStatsTxGFC is set to 0 the GFC field is ignored in test

Definition at line **1011** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtTxTransTable0\_ Struct Reference

[IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

## Data Fields

- unsigned int **phy0**:5  
[31–27] Tx Mapping value of logical phy 0
- unsigned int **phy1**:5  
[26–22] Tx Mapping value of logical phy 1
- unsigned int **phy2**:5  
[21–17] Tx Mapping value of logical phy 2
- unsigned int **reserved\_1**:1  
[16] These bits are always 0.
- unsigned int **phy3**:5  
[15–11] Tx Mapping value of logical phy 3
- unsigned int **phy4**:5  
[10–6] Tx Mapping value of logical phy 4
- unsigned int **phy5**:5  
[5–1] Tx Mapping value of logical phy 5
- unsigned int **reserved\_2**:1  
[0] These bits are always 0

---

## Detailed Description

Utopia Tx translation table Register.

Definition at line 1135 of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0\_::phy0

[31–27] Tx Mapping value of logical phy 0

Definition at line **1138** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy1
```

[26–22] Tx Mapping value of logical phy 1

Definition at line **1140** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy2
```

[21–17] Tx Mapping value of logical phy 2

Definition at line **1142** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy3
```

[15–11] Tx Mapping value of logical phy 3

Definition at line **1146** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy4
```

[10–6] Tx Mapping value of logical phy 4

Definition at line **1148** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::phy5
```

[5–1] Tx Mapping value of logical phy 5

Definition at line **1150** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::reserved_1
```

[16] These bits are always 0.

Definition at line **1144** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable0_::reserved_2
```

[0] These bits are always 0

Definition at line **1152** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtTxTransTable1\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

### Data Fields

- unsigned int **phy6**:5  
*[31–27] Tx Mapping value of logical phy 6*
- unsigned int **phy7**:5  
*[26–22] Tx Mapping value of logical phy 7*
- unsigned int **phy8**:5  
*[21–17] Tx Mapping value of logical phy 8*
- unsigned int **reserved\_1**:1  
*[16–0] These bits are always 0*
- unsigned int **phy9**:5  
*[15–11] Tx Mapping value of logical phy 3*
- unsigned int **phy10**:5  
*[10–6] Tx Mapping value of logical phy 4*
- unsigned int **phy11**:5  
*[5–1] Tx Mapping value of logical phy 5*
- unsigned int **reserved\_2**:1  
*[0] These bits are always 0*

---

### Detailed Description

Utopia Tx translation table Register.

Definition at line **1160** of file **IxAtmdAccCtrl.h**.

---

### Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1\_::phy10



[10–6] Tx Mapping value of logical phy 4

Definition at line **1173** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy11
```

[5–1] Tx Mapping value of logical phy 5

Definition at line **1175** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy6
```

[31–27] Tx Mapping value of logical phy 6

Definition at line **1163** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy7
```

[26–22] Tx Mapping value of logical phy 7

Definition at line **1165** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy8
```

[21–17] Tx Mapping value of logical phy 8

Definition at line **1167** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::phy9
```

[15–11] Tx Mapping value of logical phy 3

Definition at line **1171** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::reserved_1
```

[16–0] These bits are always 0

Definition at line **1169** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable1_::reserved_2
```

[0] These bits are always 0

Definition at line **1177** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtTxTransTable2\_ Struct Reference

[IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

## Data Fields

- unsigned int **phy12**:5  
*[31–27] Tx Mapping value of logical phy 6*
- unsigned int **phy13**:5  
*[26–22] Tx Mapping value of logical phy 7*
- unsigned int **phy14**:5  
*[21–17] Tx Mapping value of logical phy 8*
- unsigned int **reserved\_1**:1  
*[16–0] These bits are always 0*
- unsigned int **phy15**:5  
*[15–11] Tx Mapping value of logical phy 3*
- unsigned int **phy16**:5  
*[10–6] Tx Mapping value of logical phy 4*
- unsigned int **phy17**:5  
*[5–1] Tx Mapping value of logical phy 5*
- unsigned int **reserved\_2**:1  
*[0] These bits are always 0*

---

## Detailed Description

Utopia Tx translation table Register.

Definition at line **1185** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2\_::phy12

[31–27] Tx Mapping value of logical phy 6

Definition at line **1188** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy13
```

[26–22] Tx Mapping value of logical phy 7

Definition at line **1190** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy14
```

[21–17] Tx Mapping value of logical phy 8

Definition at line **1192** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy15
```

[15–11] Tx Mapping value of logical phy 3

Definition at line **1196** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy16
```

[10–6] Tx Mapping value of logical phy 4

Definition at line **1198** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::phy17
```

[5–1] Tx Mapping value of logical phy 5

Definition at line **1200** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::reserved_1
```

[16–0] These bits are always 0

Definition at line **1194** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable2_::reserved_2
```

[0] These bits are always 0

Definition at line **1202** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtTxTransTable3\_ Struct Reference

[IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

## Data Fields

- unsigned int **phy18**:5  
[31–27] Tx Mapping value of logical phy 6
- unsigned int **phy19**:5  
[26–22] Tx Mapping value of logical phy 7
- unsigned int **phy20**:5  
[21–17] Tx Mapping value of logical phy 8
- unsigned int **reserved\_1**:1  
[16–0] These bits are always 0
- unsigned int **phy21**:5  
[15–11] Tx Mapping value of logical phy 3
- unsigned int **phy22**:5  
[10–6] Tx Mapping value of logical phy 4
- unsigned int **phy23**:5  
[5–1] Tx Mapping value of logical phy 5
- unsigned int **reserved\_2**:1  
[0] These bits are always 0

---

## Detailed Description

Utopia Tx translation table Register.

Definition at line 1210 of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3\_::phy18

[31–27] Tx Mapping value of logical phy 6

Definition at line **1213** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy19
```

[26–22] Tx Mapping value of logical phy 7

Definition at line **1215** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy20
```

[21–17] Tx Mapping value of logical phy 8

Definition at line **1217** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy21
```

[15–11] Tx Mapping value of logical phy 3

Definition at line **1221** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy22
```

[10–6] Tx Mapping value of logical phy 4

Definition at line **1223** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::phy23
```

[5–1] Tx Mapping value of logical phy 5

Definition at line **1225** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::reserved_1
```

[16–0] These bits are always 0

Definition at line **1219** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable3_::reserved_2
```

[0] These bits are always 0

Definition at line **1227** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**



# IxAtmdAccUtopiaConfig::UtTxTransTable4\_ Struct Reference

[IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

## Data Fields

- unsigned int **phy24**:5  
*[31–27] Tx Mapping value of logical phy 6*
- unsigned int **phy25**:5  
*[26–22] Tx Mapping value of logical phy 7*
- unsigned int **phy26**:5  
*[21–17] Tx Mapping value of logical phy 8*
- unsigned int **reserved\_1**:1  
*[16–0] These bits are always 0*
- unsigned int **phy27**:5  
*[15–11] Tx Mapping value of logical phy 3*
- unsigned int **phy28**:5  
*[10–6] Tx Mapping value of logical phy 4*
- unsigned int **phy29**:5  
*[5–1] Tx Mapping value of logical phy 5*
- unsigned int **reserved\_2**:1  
*[0] These bits are always 0*

---

## Detailed Description

Utopia Tx translation table Register.

Definition at line **1235** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4\_::phy24

[31–27] Tx Mapping value of logical phy 6

Definition at line **1238** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy25
```

[26–22] Tx Mapping value of logical phy 7

Definition at line **1240** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy26
```

[21–17] Tx Mapping value of logical phy 8

Definition at line **1242** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy27
```

[15–11] Tx Mapping value of logical phy 3

Definition at line **1246** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy28
```

[10–6] Tx Mapping value of logical phy 4

Definition at line **1248** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::phy29
```

[5–1] Tx Mapping value of logical phy 5

Definition at line **1250** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::reserved_1
```

[16–0] These bits are always 0

Definition at line **1244** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable4_::reserved_2
```

[0] These bits are always 0

Definition at line **1252** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaConfig::UtTxTransTable5\_ Struct Reference

[IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx translation table Register.

## Data Fields

- unsigned int **phy30**:5  
*[31–27] Tx Mapping value of logical phy 6*
- unsigned int **reserved\_1**:27  
*[26–0] These bits are always 0*

---

## Detailed Description

Utopia Tx translation table Register.

Definition at line **1260** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable5\_::phy30

[31–27] Tx Mapping value of logical phy 6

Definition at line **1263** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaConfig::UtTxTransTable5\_::reserved\_1

[26–0] These bits are always 0

Definition at line **1265** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaStatus Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia status.

### Data Fields

unsigned int **utTxCellCount**  
*count of cells transmitted*

unsigned int **utTxIdleCellCount**  
*count of idle cells transmitted*

**IxAtmdAccUtopiaStatus::UtTxCellConditionStatus\_ utTxCellConditionStatus**  
*Tx cells condition status.*

unsigned int **utRxCellCount**  
*count of cell received*

unsigned int **utRxIdleCellCount**  
*count of idle cell received*

unsigned int **utRxInvalidHECount**  
*count of invalid cell received because of HEC errors*

unsigned int **utRxInvalidParCount**  
*count of invalid cell received because of parity errors*

unsigned int **utRxInvalidSizeCount**  
*count of invalid cell received because of cell size errors*

**IxAtmdAccUtopiaStatus::UtRxCellConditionStatus\_ utRxCellConditionStatus**  
*Rx cells condition status.*

---

### Detailed Description

Utopia status.

This structure is used to set/get the Utopia status parameters

- contains debug cell counters, to be accessed during a read operation

*Note:*

– the exact description of all parameters is done in the Utopia reference documents.

Definition at line **1711** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

```
struct IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_  
IxAtmdAccUtopiaStatus::utRxCellConditionStatus
```

Rx cells condition status.

```
unsigned int IxAtmdAccUtopiaStatus::utRxCellCount
```

count of cell received

Definition at line **1752** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::utRxIdleCellCount
```

count of idle cell received

Definition at line **1753** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::utRxInvalidHECount
```

count of invalid cell received because of HEC errors

Definition at line **1754** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::utRxInvalidParCount
```

count of invalid cell received because of parity errors

Definition at line **1757** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::utRxInvalidSizeCount
```

count of invalid cell received because of cell size errors

Definition at line **1760** of file **IxAtmdAccCtrl.h**.

```
struct IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_  
IxAtmdAccUtopiaStatus::utTxCellConditionStatus
```

Tx cells condition status.

```
unsigned int IxAtmdAccUtopiaStatus::utTxCellCount
```

count of cells transmitted

Definition at line **1714** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::utTxIdleCellCount
```

count of idle cells transmitted

Definition at line **1716** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaStatus::UtRxCellConditionStatus\_ Struct Reference

## [IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Rx Status Register.

### Data Fields

unsigned int **reserved\_1**:3

*[31:29] These bits are always 0.*

unsigned int **rxCellCountOvr**:1

*[28] This bit is set if the RxCellCount register overflows*

unsigned int **invalidHecCountOvr**:1

*[27] This bit is set if the InvalidHecCount register overflows.*

unsigned int **invalidParCountOvr**:1

*[26] This bit is set if the InvalidParCount register overflows.*

unsigned int **invalidSizeCountOvr**:1

*[25] This bit is set if the InvalidSizeCount register overflows.*

unsigned int **rxIdleCountOvr**:1

*[24] This bit is set if the RxIdleCount register overflows.*

unsigned int **reserved\_2**:4

*[23:20] These bits are always 0*

unsigned int **rxFIFO2Underflow**:1

*[19] This bit is set if 64-byte Receive FIFO2 indicates a FIFO underflow error condition*

unsigned int **rxFIFO1Underflow**:1

*[18] This bit is set if 64-byte Receive FIFO1 indicates a FIFO underflow error condition*

unsigned int **rxFIFO2Overflow**:1

*[17] This bit is set if 64-byte Receive FIFO2 indicates a FIFO overflow error condition*

unsigned int **rxFIFO1Overflow**:1

*[16] This bit is set if 64-byte Receive FIFO1 indicates a FIFO overflow error condition*

unsigned int **reserved\_3**:16

*[15:0] These bits are always 0*



## Detailed Description

Utopia Rx Status Register.

Definition at line **1770** of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus\_::invalidHecCountOvr

[27] This bit is set if the InvalidHecCount register overflows.

Definition at line **1775** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus\_::invalidParCountOvr

[26] This bit is set if the InvalidParCount register overflows.

Definition at line **1776** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus\_::invalidSizeCountOvr

[25] This bit is set if the InvalidSizeCount register overflows.

Definition at line **1777** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus\_::reserved\_1

[31:29] These bits are always 0.

Definition at line **1773** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus\_::reserved\_2

[23:20] These bits are always 0

Definition at line **1779** of file **IxAtmdAccCtrl.h**.

unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus\_::reserved\_3

[15:0] These bits are always 0

Definition at line **1792** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxCellCountOvr
```

[28] This bit is set if the RxCellCount register overflows

Definition at line **1774** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxFIFO1Overflow
```

[16] This bit is set if 64-byte Receive FIFO1 indicates a FIFO overflow error condition

Definition at line **1789** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxFIFO1Underflow
```

[18] This bit is set if 64-byte Receive FIFO1 indicates a FIFO underflow error condition

Definition at line **1783** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxFIFO2Overflow
```

[17] This bit is set if 64-byte Receive FIFO2 indicates a FIFO overflow error condition

Definition at line **1786** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxFIFO2Underflow
```

[19] This bit is set if 64-byte Receive FIFO2 indicates a FIFO underflow error condition

Definition at line **1780** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtRxCellConditionStatus_::rxIdleCountOvr
```

[24] This bit is set if the RxIdleCount register overflows.

Definition at line **1778** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmdAccUtopiaStatus::UtTxCellConditionStatus\_ Struct Reference

[IXP425 ATM Driver Access (IxAtmdAcc) Utopia Control API]

Utopia Tx Status Register.

## Data Fields

unsigned int **reserved\_1**:2

*[31:30] These bits are always 0*

unsigned int **txFIFO2Underflow**:1

*[29] This bit is set if 64-byte Transmit FIFO2 indicates a FIFO underflow error condition*

unsigned int **txFIFO1Underflow**:1

*[28] This bit is set if 64-byte Transmit FIFO1 indicates a FIFO underflow error condition*

unsigned int **txFIFO2Overflow**:1

*[27] This bit is set if 64-byte Transmit FIFO2 indicates a FIFO overflow error condition*

unsigned int **txFIFO1Overflow**:1

*[26] This bit is set if 64-byte Transmit FIFO1 indicates a FIFO overflow error condition*

unsigned int **txIdleCellCountOvr**:1

*[25] This bit is set if the TxIdleCellCount register overflows*

unsigned int **txCellCountOvr**:1

*[24] This bit is set if the TxCellCount register overflows*

unsigned int **reserved\_2**:24

*[23:0] These bits are always 0*

---

## Detailed Description

Utopia Tx Status Register.

Definition at line 1723 of file **IxAtmdAccCtrl.h**.

---

## Field Documentation

unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus\_::reserved\_1

[31:30] These bits are always 0

Definition at line **1726** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::reserved_2
```

[23:0] These bits are always 0

Definition at line **1749** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txCellCountOvr
```

[24] This bit is set if the TxCellCount register overflows

Definition at line **1746** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txFIFO1Overflow
```

[26] This bit is set if 64–byte Transmit FIFO1 indicates a FIFO overflow error condition

Definition at line **1739** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txFIFO1Underflow
```

[28] This bit is set if 64–byte Transmit FIFO1 indicates a FIFO underflow error condition

Definition at line **1731** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txFIFO2Overflow
```

[27] This bit is set if 64–byte Transmit FIFO2 indicates a FIFO overflow error condition

Definition at line **1735** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txFIFO2Underflow
```

[29] This bit is set if 64–byte Transmit FIFO2 indicates a FIFO underflow error condition

Definition at line **1727** of file **IxAtmdAccCtrl.h**.

```
unsigned int IxAtmdAccUtopiaStatus::UtTxCellConditionStatus_::txIdleCellCountOvr
```

[25] This bit is set if the TxIdleCellCount register overflows

Definition at line **1743** of file **IxAtmdAccCtrl.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmdAccCtrl.h**

# IxAtmmPortCfg Struct Reference

## [IXP425 ATM Manager (IxAtmm) API]

Structure contains port-specific information required to initialize IxAtmm, and specifically, the IXP425 UTOPIA Level-2 device.

### Data Fields

unsigned **reserved\_1**:11

*[31:21] Should be zero*

unsigned **UtopiaTxPhyAddr**:5

*[20:16] Address of the transmit (Tx) PHY for this port on the 5-bit UTOPIA Level-2 address bus*

unsigned **reserved\_2**:11

*[15:5] Should be zero*

unsigned **UtopiaRxPhyAddr**:5

*[4:0] Address of the receive (Rx) PHY for this port on the 5-bit UTOPIA Level-2 address bus*

---

### Detailed Description

Structure contains port-specific information required to initialize IxAtmm, and specifically, the IXP425 UTOPIA Level-2 device.

Definition at line **191** of file **IxAtmm.h**.

---

### Field Documentation

unsigned IxAtmmPortCfg::reserved\_1

*[31:21] Should be zero*

Definition at line **192** of file **IxAtmm.h**.

unsigned IxAtmmPortCfg::reserved\_2

*[15:5] Should be zero*

Definition at line **197** of file **IxAtmm.h**.

unsigned IxAtmmPortCfg::UtopiaRxPhyAddr

[4:0] Address of the receive (Rx) PHY for this port on the 5-bit UTOPIA Level-2 address bus

Definition at line **198** of file **IxAtmm.h**.

unsigned IxAtmmPortCfg::UtopiaTxPhyAddr

[20:16] Address of the transmit (Tx) PHY for this port on the 5-bit UTOPIA Level-2 address bus

Definition at line **193** of file **IxAtmm.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmm.h**

# IxAtmmVc Struct Reference

## [IXP425 ATM Manager (IxAtmm) API]

This structure describes the required attributes of a virtual connection.

### Data Fields

unsigned **vpi**  
*VPI value of this virtual connection.*

unsigned **vci**  
*VCI value of this virtual connection.*

**IxAtmmVcDirection direction**  
*VC direction.*

**IxAtmTrafficDescriptor trafficDesc**  
*Traffic descriptor of this virtual connection.*

---

### Detailed Description

This structure describes the required attributes of a virtual connection.

Definition at line **166** of file **IxAtmm.h**.

---

### Field Documentation

**IxAtmmVcDirection** IxAtmmVc::direction

VC direction.

Definition at line **169** of file **IxAtmm.h**.

**IxAtmTrafficDescriptor** IxAtmmVc::trafficDesc

Traffic descriptor of this virtual connection.

This structure is defined by the **IXP425 ATM Transmit Scheduler (IxAtmSch) API** component.

Definition at line **173** of file **IxAtmm.h**.



unsigned IxAtmmVc::vci

VCI value of this virtual connection.

Definition at line **168** of file **IxAtmm.h**.

unsigned IxAtmmVc::vpi

VPI value of this virtual connection.

Definition at line **167** of file **IxAtmm.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmm.h**

# IxAtmScheduleTable Struct Reference

## [IXP425 ATM Types (IxAtmTypes)]

This structure defines a schedule table which gives details on which data (from which VCs) should be transmitted for a forthcoming period of time for a particular port and the order in which that data should be transmitted.

## Data Fields

unsigned **tableSize**  
*Number of entries.*

unsigned **totalCellSlots**  
*Number of cells.*

**IxAtmScheduleTableEntry \* table**  
*Pointer to schedule entries.*

---

## Detailed Description

This structure defines a schedule table which gives details on which data (from which VCs) should be transmitted for a forthcoming period of time for a particular port and the order in which that data should be transmitted.

The schedule table consists of a series of entries each of which will schedule one or more cells from a particular registered VC. The total number of cells scheduled and the total number of entries in the table are also indicated.

Definition at line **384** of file **IxAtmTypes.h**.

---

## Field Documentation

**IxAtmScheduleTableEntry\*** IxAtmScheduleTable::table

Pointer to schedule entries.

Pointer to an array containing tableSize entries

Definition at line **397** of file **IxAtmTypes.h**.

unsigned IxAtmScheduleTable::tableSize

Number of entries.

Indicates the total number of entries in the table.

Definition at line **386** of file **IxAtmTypes.h**.

unsigned IxAtmScheduleTable::totalCellSlots

Number of cells.

Indicates the total number of ATM cells which are scheduled by all the entries in the table.

Definition at line **391** of file **IxAtmTypes.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmTypes.h**

# IxAtmScheduleTableEntry Struct Reference

## [IXP425 ATM Types (IxAtmTypes)]

ATM Schedule Table entry.

### Data Fields

**IxAtmConnId** **connId**

*connection Id*

unsigned int **numberOfCells**

*number of cells to transmit*

---

### Detailed Description

ATM Schedule Table entry.

This IxAtmScheduleTableEntry is used by an ATM scheduler to inform IxAtmdAcc about the data to transmit (in term of cells per VC)

This structure defines

- the number of cells to be transmitted (numberOfCells)
- the VC connection to be used for transmission (connId).

**Note:**

- When the connection Id value is IX\_ATM\_IDLE\_CELLS\_CONNID, the corresponding number of idle cells will be transmitted to the hardware.

Definition at line **352** of file **IxAtmTypes.h**.

---

### Field Documentation

**IxAtmConnId** IxAtmScheduleTableEntry::connId

connection Id

Identifier of VC from which cells are to be transmitted. When this value is IX\_ATM\_IDLE\_CELLS\_CONNID, this indicates that the system should transmit the specified number of idle cells. Unknown connIds result in the transmission of idle cells.

Definition at line **354** of file **IxAtmTypes.h**.

unsigned int IxAtmScheduleTableEntry::numberOfCells

number of cells to transmit

The number of contiguous cells to schedule from this VC at this point. The valid range is from 1 to *IX\_ATM\_SCHEDULETABLE\_MAXCELLS\_PER\_ENTRY*. This number can swap over mbufs and pdus. OverScheduling results in the transmission of idle cells.

Definition at line **362** of file **IxAtmTypes.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmTypes.h**

# IxAtmTrafficDescriptor Struct Reference

## [IXP425 ATM Types (IxAtmTypes)]

Structure describing an ATM traffic contract for a Virtual Connection (VC).

### Data Fields

**IxAtmServiceCategory atmService**

*ATM service category.*

unsigned **pcr**

*Peak Cell Rate – cells per second.*

unsigned **cdvt**

*Cell Delay Variation Tolerance – in nanoseconds.*

unsigned **scr**

*Sustained Cell Rate – cells per second.*

unsigned **mbs**

*Max Burst Size – cells.*

unsigned **mcr**

*Minimum Cell Rate – cells per second.*

unsigned **mfs**

*Max Frame Size – cells.*

---

## Detailed Description

Structure describing an ATM traffic contract for a Virtual Connection (VC).

Structure is used to specify the requested traffic contract for a VC to the IxAtmSch component using the **ixAtmSchVcModelSetup** interface.

These parameters are defined by the ATM forum working group (<http://www.atmforum.com>).

#### *Note:*

Typical values for a voice channel 64 Kbit/s

◇ atmService *IX\_ATM\_RTVBR*

◇ pcr 400 (include IP overhead, and AAL5 trailer)

◇ cdvt 5000000 (5 ms)

◇ scr = pcr

Typical values for a data channel 800 Kbit/s

- ◇ atmService *IX\_ATM\_UBR*
- ◇ pcr 1962 (include IP overhead, and AAL5 trailer)
- ◇ cdvt 5000000 (5 ms)

Definition at line **291** of file **IxAtmTypes.h**.

---

## Field Documentation

**IxAtmServiceCategory** IxAtmTrafficDescriptor::atmService

ATM service category.

Definition at line **293** of file **IxAtmTypes.h**.

**unsigned** IxAtmTrafficDescriptor::cdvt

Cell Delay Variation Tolerance – in nanoseconds.

Definition at line **295** of file **IxAtmTypes.h**.

**unsigned** IxAtmTrafficDescriptor::mbs

Max Burst Size – cells.

Definition at line **297** of file **IxAtmTypes.h**.

**unsigned** IxAtmTrafficDescriptor::mcr

Minimum Cell Rate – cells per second.

Definition at line **298** of file **IxAtmTypes.h**.

**unsigned** IxAtmTrafficDescriptor::mfs

Max Frame Size – cells.

Definition at line **299** of file **IxAtmTypes.h**.

**unsigned** IxAtmTrafficDescriptor::pcr

Peak Cell Rate – cells per second.

Definition at line **294** of file **IxAtmTypes.h**.

```
unsigned IxAtmTrafficDescriptor::scr
```

Sustained Cell Rate – cells per second.

Definition at line **296** of file **IxAtmTypes.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmTypes.h**



# IxEthAccMacAddr Struct Reference

## [IXP425 Ethernet Access (IxEthAcc) API]

The IEEE 802.3 Ethernet MAC address structure.

### Data Fields

UINT8 **macAddress** [IX\_IEEE803\_MAC\_ADDRESS\_SIZE]  
*MAC address.*

---

### Detailed Description

The IEEE 802.3 Ethernet MAC address structure.

The data should be packed with bytes xx:xx:xx:xx:xx:xx

**Note:**

The data must be packed in network byte order.

Definition at line **122** of file **IxEthAcc.h**.

---

### Field Documentation

UINT8 IxEthAccMacAddr::macAddress[IX\_IEEE803\_MAC\_ADDRESS\_SIZE]

MAC address.

Definition at line **124** of file **IxEthAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxEthAcc.h**

# IxEthDBMacAddr Struct Reference

## [IXP425 Ethernet Database (IxEthDB) API]

The IEEE 802.3 Ethernet MAC address structure.

### Data Fields

UINT8 **macAddress** [IX\_IEEE803\_MAC\_ADDRESS\_SIZE]

---

### Detailed Description

The IEEE 802.3 Ethernet MAC address structure.

The data should be packed with bytes xx:xx:xx:xx:xx:xx

**Note:**

The data must be packed in network byte order.

Definition at line **110** of file **IxEthDB.h**.

---

The documentation for this struct was generated from the following file:

- **IxEthDB.h**

# IxEthDBPortDefinition Struct Reference

## [IXP425 Ethernet Database Port Definitions (IxEthDBPortDefs)]

Port Definition – a structure contains the Port type and capabilities.

### Data Fields

**IxEthDBPortType** type  
**IxEthDBPortCapability** capabilities

---

### Detailed Description

Port Definition – a structure contains the Port type and capabilities.

Definition at line **81** of file **IxEthDBPortDefs.h**.

---

The documentation for this struct was generated from the following file:

- **IxEthDBPortDefs.h**

# IxEthEthObjStats Struct Reference

## [IXP425 Ethernet Access (IxEthAcc) API]

This struct defines the statistics returned by this component. The component returns MIB2 EthObj variables which should be obtained from the hardware or maintained by this component.

### Data Fields

UINT32 **dot3StatsAlignmentErrors**  
*link error count*

UINT32 **dot3StatsFCSErrors**  
*link error count*

UINT32 **dot3StatsFrameTooLongs**  
*link error count*

UINT32 **dot3StatsInternalMacReceiveErrors**  
*link error count*

UINT32 **LearnedEntryDiscards**  
*NPE error count.*

UINT32 **UnderflowEntryDiscards**  
*NPE error count.*

UINT32 **dot3StatsSingleCollisionFrames**  
*link error count*

UINT32 **dot3StatsMultipleCollisionFrames**  
*link error count*

UINT32 **dot3StatsDeferredTransmissions**  
*link error count*

UINT32 **dot3StatsLateCollisions**  
*link error count*

UINT32 **dot3StatsExcessiveCollisions**  
*link error count*

UINT32 **dot3StatsInternalMacTransmitErrors**  
*link error count*

UINT32 **dot3StatsCarrierSenseErrors**  
*link error count*

---

## Detailed Description

This struct defines the statistics returned by this component. The component returns MIB2 EthObj variables which should be obtained from the hardware or maintained by this component.

Definition at line **1389** of file **IxEthAcc.h**.

---

## Field Documentation

UINT32 IxEthEthObjStats::dot3StatsAlignmentErrors

link error count

Definition at line **1391** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsCarrierSenseErrors

link error count

Definition at line **1403** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsDeferredTransmissions

link error count

Definition at line **1399** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsExcessiveCollisions

link error count

Definition at line **1401** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::dot3StatsFCSErrors

link error count

Definition at line **1392** of file **IxEthAcc.h**.

#### UINT32 IxEthEthObjStats::dot3StatsFrameTooLongs

link error count

Definition at line **1393** of file **IxEthAcc.h**.

#### UINT32 IxEthEthObjStats::dot3StatsInternalMacReceiveErrors

link error count

Definition at line **1394** of file **IxEthAcc.h**.

#### UINT32 IxEthEthObjStats::dot3StatsInternalMacTransmitErrors

link error count

Definition at line **1402** of file **IxEthAcc.h**.

#### UINT32 IxEthEthObjStats::dot3StatsLateCollisions

link error count

Definition at line **1400** of file **IxEthAcc.h**.

#### UINT32 IxEthEthObjStats::dot3StatsMultipleCollisionFrames

link error count

Definition at line **1398** of file **IxEthAcc.h**.

#### UINT32 IxEthEthObjStats::dot3StatsSingleCollisionFrames

link error count

Definition at line **1397** of file **IxEthAcc.h**.

#### UINT32 IxEthEthObjStats::LearnedEntryDiscards

NPE error count.

Definition at line **1395** of file **IxEthAcc.h**.

UINT32 IxEthEthObjStats::UnderflowEntryDiscards

NPE error count.

Definition at line **1396** of file **IxEthAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxEthAcc.h**

# IxHssAccCodeletStats Struct Reference

ingroup IxHssAccCodeletCom

## Data Fields

GeneralStats gen  
ChannelisedStats chan  
PacketisedStats pkt [IX\_HSSACC\_HDLC\_PORT\_MAX]

---

## Detailed Description

ingroup IxHssAccCodeletCom

brief Type definition structure for HSS Access Codelet statistics

Definition at line **144** of file **IxHssAccCodeletCom.h**.

---

The documentation for this struct was generated from the following file:

- **IxHssAccCodeletCom.h**



# IxHssAccConfigParams Struct Reference

## [IXP425 HSS Access (IxHssAcc) API]

Structure containing HSS configuration parameters.

### Data Fields

**IxHssAccPortConfig txPortConfig**

*HSS tx port configuration.*

**IxHssAccPortConfig rxPortConfig**

*HSS rx port configuration.*

unsigned **numChannelised**

*The number of channelised timeslots (0–32).*

unsigned **hssPktChannelCount**

*The number of packetised clients (0 – 4).*

UINT8 **channelisedIdlePattern**

*The byte to be transmitted on channelised service when there is no client data to tx.*

BOOL **loopback**

*The HSS loopback state.*

unsigned **packetizedIdlePattern**

*The data to be transmitted on packetised service when there is no client data to tx.*

**IxHssAccClkSpeed clkSpeed**

*The HSS clock speed.*

---

## Detailed Description

Structure containing HSS configuration parameters.

Definition at line **568** of file **IxHssAcc.h**.

---

## Field Documentation

UINT8 IxHssAccConfigParams::channelisedIdlePattern

The byte to be transmitted on channelised service when there is no client data to tx.

Definition at line **576** of file **IxHssAcc.h**.

**IxHssAccClkSpeed** IxHssAccConfigParams::clkSpeed

The HSS clock speed.

Definition at line **583** of file **IxHssAcc.h**.

unsigned IxHssAccConfigParams::hssPktChannelCount

The number of packetised clients (0 – 4).

Definition at line **574** of file **IxHssAcc.h**.

BOOL IxHssAccConfigParams::loopback

The HSS loopback state.

Definition at line **579** of file **IxHssAcc.h**.

unsigned IxHssAccConfigParams::numChannelised

The number of channelised timeslots (0–32).

Definition at line **572** of file **IxHssAcc.h**.

unsigned IxHssAccConfigParams::packetizedIdlePattern

The data to be transmitted on packetised service when there is no client data to tx.

Definition at line **580** of file **IxHssAcc.h**.

**IxHssAccPortConfig** IxHssAccConfigParams::rxPortConfig

HSS rx port configuration.

Definition at line **571** of file **IxHssAcc.h**.

**IxHssAccPortConfig** IxHssAccConfigParams::txPortConfig

HSS tx port configuration.

Definition at line **570** of file **IxHssAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxHssAcc.h**

# IxHssAccHdlcMode Struct Reference

## [IXP425 HSS Access (IxHssAcc) API]

This structure contains 56Kbps, HDLC–mode configuration parameters.

### Data Fields

BOOL **hdlc56kMode**

*56kbps(TRUE)/64kbps(FALSE) HDLC*

**IxHssAcc56kEndianness** **hdlc56kEndian**

*56kbps data endianness*

- *ignored if hdlc56kMode is FALSE*

BOOL **hdlc56kUnusedBitPolarity0**

*The polarity '0'(TRUE)/'1'(FALSE) of the unused bit while in 56kbps mode*

- *ignored if hdlc56kMode is FALSE.*

---

## Detailed Description

This structure contains 56Kbps, HDLC–mode configuration parameters.

Definition at line **590** of file **IxHssAcc.h**.

---

## Field Documentation

**IxHssAcc56kEndianness** IxHssAccHdlcMode::hdlc56kEndian

56kbps data endianness

- ignored if hdlc56kMode is FALSE

Definition at line **593** of file **IxHssAcc.h**.

BOOL IxHssAccHdlcMode::hdlc56kMode

56kbps(TRUE)/64kbps(FALSE) HDLC

Definition at line **592** of file **IxHssAcc.h**.

BOOL IxHssAccHdlcMode::hdlc56kUnusedBitPolarity0

The polarity '0'(TRUE)/'1'(FALSE) of the unused bit while in 56kbps mode

- ignored if hdlc56kMode is FALSE.

Definition at line **595** of file **IxHssAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxHssAcc.h**

# IxHssAccPktHdlcFraming Struct Reference

## [IXP425 HSS Access (IxHssAcc) API]

This structure contains information required by the NPE to configure the HDLC co-processor.

## Data Fields

**IxHssAccPktHdlcIdleType** **hdlcIdleType**

*What to transmit when a HDLC port is idle.*

**IxHssAccBitEndian** **dataEndian**

*The HDLC data endianness.*

**IxHssAccPktCrcType** **crcType**

*The CRC type to be used for this HDLC port.*

---

## Detailed Description

This structure contains information required by the NPE to configure the HDLC co-processor.

Definition at line **605** of file **IxHssAcc.h**.

---

## Field Documentation

**IxHssAccPktCrcType** IxHssAccPktHdlcFraming::crcType

The CRC type to be used for this HDLC port.

Definition at line **609** of file **IxHssAcc.h**.

**IxHssAccBitEndian** IxHssAccPktHdlcFraming::dataEndian

The HDLC data endianness.

Definition at line **608** of file **IxHssAcc.h**.

**IxHssAccPktHdlcIdleType** IxHssAccPktHdlcFraming::hdlcIdleType

What to transmit when a HDLC port is idle.

Definition at line **607** of file **IxHssAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxHssAcc.h**

# IxHssAccPortConfig Struct Reference

## [IXP425 HSS Access (IxHssAcc) API]

Structure containing HSS port configuration parameters.

### Data Fields

**IxHssAccFrmSyncType frmSyncType**  
*frame sync pulse type (tx/rx)*

**IxHssAccFrmSyncEnable frmSyncIO**  
*how the frame sync pulse is used (tx/rx)*

**IxHssAccClkEdge frmSyncClkEdge**  
*frame sync clock edge type (tx/rx)*

**IxHssAccClkEdge dataClkEdge**  
*data clock edge type (tx/rx)*

**IxHssAccClkDir clkDirection**  
*clock direction (tx/rx)*

**IxHssAccFrmPulseUsage frmPulseUsage**  
*whether to use the frame sync pulse or not (tx/rx)*

**IxHssAccDataRate dataRate**  
*data rate in relation to the clock (tx/rx)*

**IxHssAccDataPolarity dataPolarity**  
*data polarity type (tx/rx)*

**IxHssAccBitEndian dataEndianness**  
*data endianness (tx/rx)*

**IxHssAccDrainMode drainMode**  
*tx pin open drain mode (tx)*

**IxHssAccSOFTType fBitUsage**  
*start of frame types (tx/rx)*

**IxHssAccDataEnable dataEnable**  
*whether or not to drive the data pins (tx)*

**IxHssAccTxSigType voice56kType**  
*how to drive the data pins for voice56k type (tx)*

**IxHssAccTxSigType unassignedType**



*how to drive the data pins for unassigned type (tx)*

**IxHssAccFbType fBitType**

*how to drive the Fbit (tx)*

**IxHssAcc56kEndianness voice56kEndian**

*56k data endianness when using the 56k type (tx)*

**IxHssAcc56kSel voice56kSel**

*56k data transmission type when using the 56k type (tx)*

unsigned **frmOffset**

*frame pulse offset in bits wrt the first timeslot (0–1023) (tx/rx)*

unsigned **maxFrmSize**

*frame size in bits (1–1024) (tx/rx)*

---

## Detailed Description

Structure containing HSS port configuration parameters.

Note: All of these are used for TX. Only some are specific to RX.

Definition at line **530** of file **IxHssAcc.h**.

---

## Field Documentation

**IxHssAccClkDir** IxHssAccPortConfig::clkDirection

clock direction (tx/rx)

Definition at line **538** of file **IxHssAcc.h**.

**IxHssAccClkEdge** IxHssAccPortConfig::dataClkEdge

data clock edge type (tx/rx)

Definition at line **537** of file **IxHssAcc.h**.

**IxHssAccDataEnable** IxHssAccPortConfig::dataEnable

whether or not to drive the data pins (tx)

Definition at line **547** of file **IxHssAcc.h**.

**IxHssAccBitEndian** IxHssAccPortConfig::dataEndianness

data endianness (tx/rx)

Definition at line **544** of file **IxHssAcc.h**.

**IxHssAccDataPolarity** IxHssAccPortConfig::dataPolarity

data polarity type (tx/rx)

Definition at line **543** of file **IxHssAcc.h**.

**IxHssAccDataRate** IxHssAccPortConfig::dataRate

data rate in relation to the clock (tx/rx)

Definition at line **541** of file **IxHssAcc.h**.

**IxHssAccDrainMode** IxHssAccPortConfig::drainMode

tx pin open drain mode (tx)

Definition at line **545** of file **IxHssAcc.h**.

**IxHssAccFbType** IxHssAccPortConfig::fBitType

how to drive the Fbit (tx)

Definition at line **553** of file **IxHssAcc.h**.

**IxHssAccSOFTType** IxHssAccPortConfig::fBitUsage

start of frame types (tx/rx)

Definition at line **546** of file **IxHssAcc.h**.

unsigned IxHssAccPortConfig::frmOffset

frame pulse offset in bits wrt the first timeslot (0–1023) (tx/rx)

Definition at line **558** of file **IxHssAcc.h**.

**IxHssAccFrmPulseUsage** IxHssAccPortConfig::frmPulseUsage

whether to use the frame sync pulse or not (tx/rx)

Definition at line **539** of file **IxHssAcc.h**.

**IxHssAccClkEdge** IxHssAccPortConfig::frmSyncClkEdge

frame sync clock edge type (tx/rx)

Definition at line **535** of file **IxHssAcc.h**.

**IxHssAccFrmSyncEnable** IxHssAccPortConfig::frmSyncIO

how the frame sync pulse is used (tx/rx)

Definition at line **533** of file **IxHssAcc.h**.

**IxHssAccFrmSyncType** IxHssAccPortConfig::frmSyncType

frame sync pulse type (tx/rx)

Definition at line **532** of file **IxHssAcc.h**.

unsigned IxHssAccPortConfig::maxFrmSize

frame size in bits (1–1024) (tx/rx)

Definition at line **560** of file **IxHssAcc.h**.

**IxHssAccTxSigType** IxHssAccPortConfig::unassignedType

how to drive the data pins for unassigned type (tx)

Definition at line **551** of file **IxHssAcc.h**.

**IxHssAcc56kEndianness** IxHssAccPortConfig::voice56kEndian

56k data endianness when using the 56k type (tx)

Definition at line **554** of file **IxHssAcc.h**.

**IxHssAcc56kSel** IxHssAccPortConfig::voice56kSel

56k data transmission type when using the 56k type (tx)

Definition at line **556** of file **IxHssAcc.h**.

**IxHssAccTxSigType** IxHssAccPortConfig::voice56kType

how to drive the data pins for voice56k type (tx)

Definition at line **549** of file **IxHssAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxHssAcc.h**

# IxMbufPool Struct Reference

## [IXP425 OS Memory Buffer Pool Management (IxOsBuffPoolMgt) API]

Implementation of buffer pool structure for use with non-VxWorks OS.

### Data Fields

**IX\_MBUF \* nextFreeBuf**  
*Pointer to the next free mbuf.*

**void \* mbufMemPtr**  
*Pointer to the mbuf memory area.*

**void \* dataMemPtr**  
*Pointer to the data memory area.*

**int bufDataSize**  
*The size of the data portion of each mbuf.*

**int totalBufsInPool**  
*Total number of mbufs in the pool.*

**int freeBufsInPool**  
*Number of free mbufs currently in the pool.*

**int mbufMemSize**  
*The size of the pool mbuf memory area.*

**int dataMemSize**  
*The size of the pool data memory area.*

**char name [IX\_MBUF\_POOL\_NAME\_LEN+1]**  
*Descriptive name for pool.*

**IxMbufPoolAllocationType poolAllocType**

---

### Detailed Description

Implementation of buffer pool structure for use with non-VxWorks OS.

Definition at line 268 of file **IxOsBuffPoolMgt.h**.

---

# Field Documentation

`int IxMbufPool::bufDataSize`

The size of the data portion of each mbuf.

Definition at line **273** of file **IxOsBuffPoolMgt.h**.

`void* IxMbufPool::dataMemPtr`

Pointer to the data memory area.

Definition at line **272** of file **IxOsBuffPoolMgt.h**.

`int IxMbufPool::dataMemSize`

The size of the pool data memory area.

Definition at line **277** of file **IxOsBuffPoolMgt.h**.

`int IxMbufPool::freeBufsInPool`

Number of free mbufs currently in the pool.

Definition at line **275** of file **IxOsBuffPoolMgt.h**.

`void* IxMbufPool::mbufMemPtr`

Pointer to the mbuf memory area.

Definition at line **271** of file **IxOsBuffPoolMgt.h**.

`int IxMbufPool::mbufMemSize`

The size of the pool mbuf memory area.

Definition at line **276** of file **IxOsBuffPoolMgt.h**.

`char IxMbufPool::name[IX_MBUF_POOL_NAME_LEN + 1]`

Descriptive name for pool.

Definition at line **278** of file **IxOsBuffPoolMgt.h**.

**IX\_MBUF\*** IxMbufPool::nextFreeBuf

Pointer to the next free mbuf.

Definition at line **270** of file **IxOsBuffPoolMgt.h**.

**int** IxMbufPool::totalBufsInPool

Total number of mbufs in the pool.

Definition at line **274** of file **IxOsBuffPoolMgt.h**.

---

The documentation for this struct was generated from the following file:

- **IxOsBuffPoolMgt.h**

# IxNpeA\_NpePacketDescriptor Struct Reference

## [IXP425 NPE-A (IxNpeA) API]

HSS Packetized NpePacket Descriptor Structure.

### Data Fields

UINT8 **status**

*Status of the packet passed to the client.*

UINT8 **errorCount**

*Number of errors.*

UINT8 **chainCount**

*Mbuf chain count e.g.*

UINT8 **rsvdByte0**

*Reserved byte to make the descriptor word align.*

UINT16 **packetLength**

*Packet Length.*

UINT16 **rsvdShort0**

*Reserved short to make the descriptor a word align.*

IX\_MBUF \* **pRootMbuf**

*Pointer to Root mbuf.*

IX\_MBUF \* **pNextMbuf**

*Pointer to next mbuf.*

UINT8 \* **pMbufData**

*Pointer to the current mbuf->data.*

UINT32 **mbufLength**

*Current mbuf length.*

---

### Detailed Description

HSS Packetized NpePacket Descriptor Structure.

Definition at line **1080** of file **IxNpeA.h**.

---



# Field Documentation

## UINT8 IxNpeA\_NpePacketDescriptor::chainCount

Mbuf chain count e.g.

0 – No mbuf chain

Definition at line **1084** of file **IxNpeA.h**.

## UINT8 IxNpeA\_NpePacketDescriptor::errorCount

Number of errors.

Definition at line **1083** of file **IxNpeA.h**.

## UINT32 IxNpeA\_NpePacketDescriptor::mbufLength

Current mbuf length.

Definition at line **1093** of file **IxNpeA.h**.

## UINT16 IxNpeA\_NpePacketDescriptor::packetLength

Packet Length.

Definition at line **1087** of file **IxNpeA.h**.

## UINT8\* IxNpeA\_NpePacketDescriptor::pMbufData

Pointer to the current mbuf->data.

Definition at line **1092** of file **IxNpeA.h**.

## IX\_MBUF\* IxNpeA\_NpePacketDescriptor::pNextMbuf

Pointer to next mbuf.

Definition at line **1091** of file **IxNpeA.h**.

## IX\_MBUF\* IxNpeA\_NpePacketDescriptor::pRootMbuf

Pointer to Root mbuf.

Definition at line **1090** of file **IxNpeA.h**.

UINT8 IxNpeA\_NpePacketDescriptor::rsvdByte0

Reserved byte to make the descriptor word align.

Definition at line **1085** of file **IxNpeA.h**.

UINT16 IxNpeA\_NpePacketDescriptor::rsvdShort0

Reserved short to make the descriptor a word align.

Definition at line **1088** of file **IxNpeA.h**.

UINT8 IxNpeA\_NpePacketDescriptor::status

Status of the packet passed to the client.

Definition at line **1082** of file **IxNpeA.h**.

---

The documentation for this struct was generated from the following file:

- **IxNpeA.h**

# IxNpeA\_RxAtmVc Struct Reference

## [IXP425 NPE-A (IxNpeA) API]

Rx Descriptor definition.

### Data Fields

UINT32 **rxBitField**  
*Recieved bit field.*

UINT32 **atmCellHeader**  
*ATM Cell Header.*

UINT32 **rsvdWord0**  
*Reserved field.*

UINT16 **currMbufLen**  
*Mbuf Length.*

UINT8 **timeLimit**  
*Payload Reassembly timeLimit (used for aal0\_xx only).*

UINT8 **rsvdByte0**  
*Reserved field.*

UINT32 **rsvdWord1**  
*Reserved field.*

IX\_MBUF \* **pCurrMbuf**  
*Pointer to current mbuf.*

unsigned char \* **pCurrMbufData**  
*Pointer to current mbuf->data.*

IX\_MBUF \* **pNextMbuf**  
*Pointer to next mbuf.*

UINT32 **totalLen**  
*Total Length.*

UINT32 **aal5CrcResidue**  
*AAL5 CRC Residue.*

## Detailed Description

Rx Descriptor definition.

Definition at line **1039** of file **IxNpeA.h**.

---

## Field Documentation

UINT32 IxNpeA\_RxAtmVc::aal5CrcResidue

AAL5 CRC Residue.

Definition at line **1052** of file **IxNpeA.h**.

UINT32 IxNpeA\_RxAtmVc::atmCellHeader

ATM Cell Header.

Definition at line **1042** of file **IxNpeA.h**.

UINT16 IxNpeA\_RxAtmVc::currMbufLen

Mbuf Length.

Definition at line **1044** of file **IxNpeA.h**.

IX\_MBUF\* IxNpeA\_RxAtmVc::pCurrMbuf

Pointer to current mbuf.

Definition at line **1048** of file **IxNpeA.h**.

unsigned char\* IxNpeA\_RxAtmVc::pCurrMbufData

Pointer to current mbuf->data.

Definition at line **1049** of file **IxNpeA.h**.

IX\_MBUF\* IxNpeA\_RxAtmVc::pNextMbuf

Pointer to next mbuf.

Definition at line **1050** of file **IxNpeA.h**.

UINT8 IxNpeA\_RxAtmVc::rsvdByte0

Reserved field.

Definition at line **1046** of file **IxNpeA.h**.

UINT32 IxNpeA\_RxAtmVc::rsvdWord0

Reserved field.

Definition at line **1043** of file **IxNpeA.h**.

UINT32 IxNpeA\_RxAtmVc::rsvdWord1

Reserved field.

Definition at line **1047** of file **IxNpeA.h**.

UINT32 IxNpeA\_RxAtmVc::rxBitField

Recieved bit field.

Definition at line **1041** of file **IxNpeA.h**.

UINT8 IxNpeA\_RxAtmVc::timeLimit

Payload Reassembly timeLimit (used for aal0\_xx only).

Definition at line **1045** of file **IxNpeA.h**.

UINT32 IxNpeA\_RxAtmVc::totalLen

Total Length.

Definition at line **1051** of file **IxNpeA.h**.

---

The documentation for this struct was generated from the following file:

- **IxNpeA.h**

# IxNpeA\_TxAtmVc Struct Reference

## [IXP425 NPE–A (IxNpeA) API]

Tx Descriptor definition.

### Data Fields

UINT8 **port**  
*Tx Port number.*

UINT8 **aalType**  
*AAL Type.*

UINT16 **currMbufLen**  
*mbuf length*

UINT32 **atmCellHeader**  
*ATM cell header.*

IX\_MBUF \* **pCurrMbuf**  
*pointer to mbuf*

unsigned char \* **pCurrMbufData**  
*Pointer to mbuf->dat.*

IX\_MBUF \* **pNextMbuf**  
*Pointer to next mbuf.*

UINT32 **totalLen**  
*Total Length.*

UINT32 **aal5CrcResidue**  
*AAL5 CRC Residue.*

---

### Detailed Description

Tx Descriptor definition.

Definition at line **1017** of file **IxNpeA.h**.

---

### Field Documentation

UINT32 IxNpeA\_TxAtmVc::aal5CrcResidue

AAL5 CRC Residue.

Definition at line **1027** of file **IxNpeA.h**.

UINT8 IxNpeA\_TxAtmVc::aalType

AAL Type.

Definition at line **1020** of file **IxNpeA.h**.

UINT32 IxNpeA\_TxAtmVc::atmCellHeader

ATM cell header.

Definition at line **1022** of file **IxNpeA.h**.

UINT16 IxNpeA\_TxAtmVc::currMbufLen

mbuf length

Definition at line **1021** of file **IxNpeA.h**.

IX\_MBUF\* IxNpeA\_TxAtmVc::pCurrMbuf

pointer to mbuf

Definition at line **1023** of file **IxNpeA.h**.

unsigned char\* IxNpeA\_TxAtmVc::pCurrMbufData

Pointer to mbuf->dat.

Definition at line **1024** of file **IxNpeA.h**.

IX\_MBUF\* IxNpeA\_TxAtmVc::pNextMbuf

Pointer to next mbuf.

Definition at line **1025** of file **IxNpeA.h**.

UINT8 IxNpeA\_TxAtmVc::port

Tx Port number.

Definition at line **1019** of file **IxNpeA.h**.

UINT32 IxNpeA\_TxAtmVc::totalLen

Total Length.

Definition at line **1026** of file **IxNpeA.h**.

---

The documentation for this struct was generated from the following file:

- **IxNpeA.h**



# IxNpeDIImageId Struct Reference

## [IXP425 NPE–Downloader (IxNpeDI) API]

Image Id to identify each image contained in an image library.

### Data Fields

**IxNpeDI****NpeId** **npeId**  
*NPE ID.*

**IxNpeDI****FunctionalityId** **functionalityId**  
*Build ID indicates functionality of image.*

**IxNpeDI****Major** **major**  
*Major Release Number.*

**IxNpeDI****Minor** **minor**  
*Minor Revision Number.*

---

## Detailed Description

Image Id to identify each image contained in an image library.

### *Warning:*

**THIS struct HAS BEEN DEPRECATED AND SHOULD NOT BE USED.** It will be removed in a future release. See **ixNpeDINpeInitAndStart** for more information.

Definition at line **642** of file **IxNpeDI.h**.

---

## Field Documentation

**IxNpeDI****FunctionalityId** IxNpeDIImageId::functionalityId

Build ID indicates functionality of image.

Definition at line **645** of file **IxNpeDI.h**.

**IxNpeDI****Major** IxNpeDIImageId::major

Major Release Number.

Definition at line **646** of file **IxNpeDI.h**.

**IxNpeDIMinor** IxNpeDIImageId::minor

Minor Revision Number.

Definition at line **647** of file **IxNpeDI.h**.

**IxNpeDINpeId** IxNpeDIImageId::npeId

NPE ID.

Definition at line **644** of file **IxNpeDI.h**.

---

The documentation for this struct was generated from the following file:

- **IxNpeDI.h**

# IxNpeMhMessage Struct Reference

## [IXP425 NPE Message Handler (IxNpeMh) API]

The 2–word message structure to send to and receive from the NPEs.

### Data Fields

UINT32 **data** [2]

*the actual data of the message*

---

### Detailed Description

The 2–word message structure to send to and receive from the NPEs.

Definition at line **109** of file **IxNpeMh.h**.

---

### Field Documentation

UINT32 IxNpeMhMessage::data[2]

the actual data of the message

Definition at line **111** of file **IxNpeMh.h**.

---

The documentation for this struct was generated from the following file:

- **IxNpeMh.h**

# IxOamITU610Cell Struct Reference

OAM ITU610 Cell.

## Data Fields

`atmCellHeader` header  
`IxOamITU610Payload` payload

---

## Detailed Description

OAM ITU610 Cell.

Definition at line **240** of file **IxAtmCodelet\_p.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmCodelet\_p.h**

# IxOamITU610GenericPayload Struct Reference

Generic payload isn't a real payload but is used for checking which payload type a received OAM cell is.

## Data Fields

UINT8 **oamTypeAndFunction**

UINT8 **reserved** [IX\_OAM\_ITU610\_GENERIC\_PAYLOAD\_RESERVED\_BYTES\_LEN]

UINT8 **reservedAndCrc10** [IX\_OAM\_ITU610\_RESERVED\_AND\_CRC10\_LEN]

---

## Detailed Description

Generic payload isn't a real payload but is used for checking which payload type a received OAM cell is.

Definition at line **220** of file **IxAtmCodelet\_p.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmCodelet\_p.h**

# IxOamITU610LbPayload Struct Reference

Oam cells payload typedefs.

## Data Fields

UINT8 **oamTypeAndFunction**  
UINT8 **loopbackIndication**  
UINT8 **correlationTag** [IX\_OAM\_ITU610\_LB\_CORRELATION\_TAG\_LEN]  
UINT8 **llid** [IX\_OAM\_ITU610\_LOCATION\_ID\_LEN]  
UINT8 **sourceId** [IX\_OAM\_ITU610\_LOCATION\_ID\_LEN]  
UINT8 **reserved** [IX\_OAM\_ITU610\_LB\_RESERVED\_BYTES\_LEN]  
UINT8 **reservedAndCrc10** [IX\_OAM\_ITU610\_RESERVED\_AND\_CRC10\_LEN]

---

## Detailed Description

Oam cells payload typedefs.

Definition at line **205** of file **IxAtmCodelet\_p.h**.

---

The documentation for this struct was generated from the following file:

- **IxAtmCodelet\_p.h**

# IxOamITU610Payload Union Reference

OAM ITU610 Payload.

## Data Fields

**IxOamITU610LbPayload** lbPayload  
**IxOamITU610GenericPayload** genericPayload

---

## Detailed Description

OAM ITU610 Payload.

Definition at line **230** of file **IxAtmCodelet\_p.h**.

---

The documentation for this union was generated from the following file:

- **IxAtmCodelet\_p.h**

# IxPerfProfAccBusPmuResults Struct Reference

## [IXP425 Performance Profiling (IxPerfProfAcc) API]

Results obtained from running the Bus Pmu component. The results are obtained when the get functions is called.

## Data Fields

UINT32 **statsToGetLower27Bit** [IX\_PERFPROF\_ACC\_BUS\_PMU\_MAX\_PECs]  
*Lower 27 Bit of counter value.*

UINT32 **statsToGetUpper32Bit** [IX\_PERFPROF\_ACC\_BUS\_PMU\_MAX\_PECs]  
*Upper 32 Bit of counter value.*

---

## Detailed Description

Results obtained from running the Bus Pmu component. The results are obtained when the get functions is called.

Definition at line **189** of file **IxPerfProfAcc.h**.

---

## Field Documentation

UINT32  
IxPerfProfAccBusPmuResults::statsToGetLower27Bit[IX\_PERFPROF\_ACC\_BUS\_PMU\_MAX\_PECs]

Lower 27 Bit of counter value.

Definition at line **191** of file **IxPerfProfAcc.h**.

UINT32  
IxPerfProfAccBusPmuResults::statsToGetUpper32Bit[IX\_PERFPROF\_ACC\_BUS\_PMU\_MAX\_PECs]

Upper 32 Bit of counter value.

Definition at line **192** of file **IxPerfProfAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxPerfProfAcc.h**



# IxPerfProfAccXcycleResults Struct Reference

## [IXP425 Performance Profiling (IxPerfProfAcc) API]

Results obtained from Xcycle run.

### Data Fields

float **maxIdlePercentage**  
*maximum percentage of Idle cycles*

float **minIdlePercentage**  
*minimum percentage of Idle cycles*

float **aveIdlePercentage**  
*average percentage of Idle cycles*

UINT32 **totalMeasurements**  
*total number of measurement made*

---

### Detailed Description

Results obtained from Xcycle run.

Definition at line **175** of file **IxPerfProfAcc.h**.

---

### Field Documentation

float IxPerfProfAccXcycleResults::aveIdlePercentage

average percentage of Idle cycles

Definition at line **179** of file **IxPerfProfAcc.h**.

float IxPerfProfAccXcycleResults::maxIdlePercentage

maximum percentage of Idle cycles

Definition at line **177** of file **IxPerfProfAcc.h**.

float IxPerfProfAccXcycleResults::minIdlePercentage

minimum percentage of Idle cycles

Definition at line **178** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXcycleResults::totalMeasurements

total number of measurement made

Definition at line **180** of file **IxPerfProfAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxPerfProfAcc.h**

# IxPerfProfAccXscalePmuEvtCnt Struct Reference

## [IXP425 Performance Profiling (IxPerfProfAcc) API]

contains results of a counter

### Data Fields

UINT32 **lower32BitsEventCount**

*lower 32bits value of the event counter*

UINT32 **upper32BitsEventCount**

*upper 32bits value of the event counter*

---

## Detailed Description

contains results of a counter

Structure contains the results of a counter, which are split into the lower and upper 32 bits of the final count

Definition at line **145** of file **IxPerfProfAcc.h**.

---

## Field Documentation

UINT32 IxPerfProfAccXscalePmuEvtCnt::lower32BitsEventCount

lower 32bits value of the event counter

Definition at line **147** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuEvtCnt::upper32BitsEventCount

upper 32bits value of the event counter

Definition at line **148** of file **IxPerfProfAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxPerfProfAcc.h**

# IxPerfProfAccXscalePmuResults Struct Reference

## [IXP425 Performance Profiling (IxPerfProfAcc) API]

contains results of counters and their overflow

### Data Fields

UINT32 **clk\_value**

*current value of clock counter*

UINT32 **clk\_samples**

*number of clock counter overflows*

UINT32 **event1\_value**

*current value of event 1 counter*

UINT32 **event1\_samples**

*number of event 1 counter overflows*

UINT32 **event2\_value**

*current value of event 2 counter*

UINT32 **event2\_samples**

*number of event 2 counter overflows*

UINT32 **event3\_value**

*current value of event 3 counter*

UINT32 **event3\_samples**

*number of event 3 counter overflows*

UINT32 **event4\_value**

*current value of event 4 counter*

UINT32 **event4\_samples**

*number of event 4 counter overflows*

---

### Detailed Description

contains results of counters and their overflow

Structure contains all values of counters and associated overflows. The specific event and clock counters are determined by the user

Definition at line **157** of file **IxPerfProfAcc.h**.

---

## Field Documentation

UINT32 IxPerfProfAccXscalePmuResults::clk\_samples

number of clock counter overflows

Definition at line **160** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuResults::clk\_value

current value of clock counter

Definition at line **159** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuResults::event1\_samples

number of event 1 counter overflows

Definition at line **162** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuResults::event1\_value

current value of event 1 counter

Definition at line **161** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuResults::event2\_samples

number of event 2 counter overflows

Definition at line **164** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuResults::event2\_value

current value of event 2 counter

Definition at line **163** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuResults::event3\_samples

number of event 3 counter overflows

Definition at line **166** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuResults::event3\_value

current value of event 3 counter

Definition at line **165** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuResults::event4\_samples

number of event 4 counter overflows

Definition at line **168** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuResults::event4\_value

current value of event 4 counter

Definition at line **167** of file **IxPerfProfAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxPerfProfAcc.h**

# IxPerfProfAccXscalePmuSamplePcProfile Struct Reference

## [IXP425 Performance Profiling (IxPerfProfAcc) API]

contains summary of samples taken

### Data Fields

UINT32 **programCounter**

*the program counter value of the sample*

UINT32 **freq**

*the frequency of the occurrence of the sample*

---

## Detailed Description

contains summary of samples taken

Structure contains all details of each program counter value – frequency that PC occurs

Definition at line **133** of file **IxPerfProfAcc.h**.

---

## Field Documentation

UINT32 IxPerfProfAccXscalePmuSamplePcProfile::freq

the frequency of the occurrence of the sample

Definition at line **136** of file **IxPerfProfAcc.h**.

UINT32 IxPerfProfAccXscalePmuSamplePcProfile::programCounter

the program counter value of the sample

Definition at line **135** of file **IxPerfProfAcc.h**.

---

The documentation for this struct was generated from the following file:

- **IxPerfProfAcc.h**

# IxQMgrQInlinedReadWriteInfo Struct Reference

## [IXP425 Queue Manager (IxQMgr) API]

Internal structure to facilitate inlining functions in **IxQMgr.h**.

### Data Fields

UINT32 **qOflowStatBitMask**  
*overflow status mask*

UINT32 **qWriteCount**  
*queue write count*

volatile UINT32 \* **qAccRegAddr**  
*access register*

volatile UINT32 \* **qUOStatRegAddr**  
*status register*

volatile UINT32 \* **qConfigRegAddr**  
*config register*

UINT32 **qEntrySizeInWords**  
*queue entry size in words*

UINT32 **qSizeInEntries**  
*queue size in entries*

UINT32 **qUflowStatBitMask**  
*underflow status mask*

UINT32 **qReadCount**  
*queue read count*

---

### Detailed Description

Internal structure to facilitate inlining functions in **IxQMgr.h**.

Definition at line **979** of file **IxQMgr.h**.

---

### Field Documentation

```
volatile UINT32* IxQMgrQInlinedReadWriteInfo::qAccRegAddr
```



access register

Definition at line **986** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstRead()**, **ixQMgrQBurstWrite()**, **ixQMgrQRead()**, and **ixQMgrQWrite()**.

volatile UINT32\* IxQMgrQInlinedReadWriteInfo::qConfigRegAddr

config register

Definition at line **988** of file **IxQMgr.h**.

Referenced by **ixQMgrQRead()**, and **ixQMgrQWrite()**.

UINT32 IxQMgrQInlinedReadWriteInfo::qEntrySizeInWords

queue entry size in words

Definition at line **989** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstRead()**, **ixQMgrQBurstWrite()**, **ixQMgrQRead()**, and **ixQMgrQWrite()**.

UINT32 IxQMgrQInlinedReadWriteInfo::qOflowStatBitMask

overflow status mask

Definition at line **982** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstWrite()**, and **ixQMgrQWrite()**.

UINT32 IxQMgrQInlinedReadWriteInfo::qReadCount

queue read count

Definition at line **994** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstRead()**, and **ixQMgrQRead()**.

UINT32 IxQMgrQInlinedReadWriteInfo::qSizeInEntries

queue size in entries

Definition at line **990** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstWrite()**, **ixQMgrQRead()**, and **ixQMgrQWrite()**.

UINT32 IxQMgrQInlinedReadWriteInfo::qUflowStatBitMask

underflow status mask

Definition at line **993** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstRead()**, and **ixQMgrQRead()**.

volatile UINT32\* IxQMgrQInlinedReadWriteInfo::qUOStatRegAddr

status register

Definition at line **987** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstRead()**, **ixQMgrQBurstWrite()**, **ixQMgrQRead()**, and **ixQMgrQWrite()**.

UINT32 IxQMgrQInlinedReadWriteInfo::qWriteCount

queue write count

Definition at line **983** of file **IxQMgr.h**.

Referenced by **ixQMgrQBurstWrite()**, and **ixQMgrQWrite()**.

---

The documentation for this struct was generated from the following file:

- **IxQMgr.h**

# ixUARTDev Struct Reference

## [IXP425 UART Access (IxUARTAcc) API, IXP425 UART Access (IxUARTAcc) API]

Device descriptor for the UART.

### Data Fields

<code>UINT8 * <b>addr</b></code>	<i>device base address</i>
<code>ixUARTMode <b>mode</b></code>	<i>interrupt, polled or loopback</i>
<code>int <b>baudRate</b></code>	<i>baud rate</i>
<code>int <b>freq</b></code>	<i>UART clock frequency.</i>
<code>int <b>options</b></code>	<i>hardware options</i>
<code>int <b>fifoSize</b></code>	<i>FIFO xmit size.</i>
<code>ixUARTStats <b>stats</b></code>	<i>device statistics</i>

---

### Detailed Description

Device descriptor for the UART.

Definition at line **342** of file **IxUART.h**.

---

### Field Documentation

`UINT8* ixUARTDev::addr`

device base address

Definition at line **344** of file **IxUART.h**.

int ixUARTDev::baudRate

baud rate

Definition at line **346** of file **IxUART.h**.

int ixUARTDev::fifoSize

FIFO xmit size.

Definition at line **349** of file **IxUART.h**.

int ixUARTDev::freq

UART clock frequency.

Definition at line **347** of file **IxUART.h**.

**ixUARTMode** ixUARTDev::mode

interrupt, polled or loopback

Definition at line **345** of file **IxUART.h**.

int ixUARTDev::options

hardware options

Definition at line **348** of file **IxUART.h**.

**ixUARTStats** ixUARTDev::stats

device statistics

Definition at line **351** of file **IxUART.h**.

---

The documentation for this struct was generated from the following file:

- **IxUART.h**

# ixUARTStats Struct Reference

[IXP425 UART Access (IxUARTAcc) API, IXP425 UART Access (IxUARTAcc) API]

Statistics for the UART.

## Data Fields

UINT32 **rxCount**  
UINT32 **txCount**  
UINT32 **overrunErr**  
UINT32 **parityErr**  
UINT32 **framingErr**  
UINT32 **breakErr**

---

## Detailed Description

Statistics for the UART.

Definition at line **328** of file **IxUART.h**.

---

The documentation for this struct was generated from the following file:

- **IxUART.h**

# PacketisedStats Struct Reference

ingroup IxHssAccCodeletCom

## Data Fields

UINT32 **txPackets**  
UINT32 **txBytes**  
UINT32 **txNoBuffers**  
UINT32 **rxPackets**  
UINT32 **rxBytes**  
UINT32 **rxNoBuffers**  
UINT32 **rxIdles**  
UINT32 **rxVerifyFails**  
UINT32 **connectFails**  
UINT32 **portEnableFails**  
UINT32 **txFails**  
UINT32 **replenishFails**  
UINT32 **portDisableFails**  
UINT32 **disconnectFails**  
UINT32 **txBufsInUse**  
UINT32 **rxBufsInUse**  
UINT32 **stopShutdownErrors**  
UINT32 **hdlcAlignErrors**  
UINT32 **hdlcFcsErrors**  
UINT32 **rxQueueEmptyErrors**  
UINT32 **hdlcMaxSizeErrors**  
UINT32 **hdlcAbortErrors**  
UINT32 **disconnectErrors**  
UINT32 **unrecognisedErrors**

---

## Detailed Description

ingroup IxHssAccCodeletCom

brief Type definition structure for Packetised statistics

Definition at line 111 of file **IxHssAccCodeletCom.h**.

---

The documentation for this struct was generated from the following file:

- **IxHssAccCodeletCom.h**

# USBDevice Struct Reference

## [IXP425 USB Driver Public API]

USBDevice.

### Data Fields

UINT32 **baseIOAddress**

*base I/O device address*

UINT32 **interruptLevel**

*device IRQ*

UINT32 **lastError**

*detailed error of last function call*

UINT32 **deviceIndex**

*USB device index.*

UINT32 **flags**

*initialization flags*

UINT8 **deviceContext** [USB\_CONTEXT\_SIZE]

*used by the driver to identify the device*

---

## Detailed Description

USBDevice.

Definition at line **74** of file **usbtypes.h**.

---

## Field Documentation

UINT32 USBDevice::baseIOAddress

base I/O device address

Definition at line **76** of file **usbtypes.h**.

UINT8 USBDevice::deviceContext[USB\_CONTEXT\_SIZE]

used by the driver to identify the device

Definition at line **81** of file **usbtypes.h**.

#### UINT32 USBDevice::deviceIndex

USB device index.

Definition at line **79** of file **usbtypes.h**.

#### UINT32 USBDevice::flags

initialization flags

Definition at line **80** of file **usbtypes.h**.

#### UINT32 USBDevice::interruptLevel

device IRQ

Definition at line **77** of file **usbtypes.h**.

#### UINT32 USBDevice::lastError

detailed error of last function call

Definition at line **78** of file **usbtypes.h**.

---

The documentation for this struct was generated from the following file:

- **usbtypes.h**



# USBDeviceCounters Struct Reference

file `usbprivatetypes.h`

## Data Fields

UINT32 **frames**  
UINT32 **irqCount**  
UINT32 **Rx**  
UINT32 **Tx**  
UINT32 **DRx**  
UINT32 **DTx**  
UINT32 **bytesRx**  
UINT32 **bytesTx**  
UINT32 **setup**

---

## Detailed Description

file `usbprivatetypes.h`

author Intel Corporation date 30-OCT-2001

This file contains the private USB Driver data types

File Version:

### *Revision*

1.13

— Intel Copyright Notice —

Copyright 2001–2003 Intel Corporation All Rights Reserved.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers or licensors. Title to the Material remains with Intel Corporation or its suppliers and licensors.

The Material is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way except in accordance with the applicable license agreement .

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel, except in accordance with the applicable license agreement.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.

For further details, please see the file README.TXT distributed with this software.

— End Intel Copyright Notice —

Definition at line **50** of file **usbprivatetypes.h**.

---

The documentation for this struct was generated from the following file:

- **usbprivatetypes.h**

# USBSetupPacket Struct Reference

## [IXP425 USB Driver Public API]

Standard USB Setup packet components, see the USB Specification 1.1.

### Data Fields

UCHAR **bmRequestType**

UCHAR **bRequest**

UINT16 **wValue**

UINT16 **wIndex**

UINT16 **wLength**

---

### Detailed Description

Standard USB Setup packet components, see the USB Specification 1.1.

Definition at line **60** of file **usbstd.h**.

---

The documentation for this struct was generated from the following file:

- **usbstd.h**