Working DraftProject American National Standard

T11/1841-D

revision 0.82
5 February 2013

Information technology - Storage Management - HBA - 2nd Generation (SM-HBA-2)

T11 Technical Editor:

Heather M Lanigan
NetApp
7301 Kit Creek Rd
Research Triangle Park, NC 27709

USA

Email: heather.lanigan@netapp.com

Points of Contact

InterNational Committee for Information Technology Standards (INCITS) T11 Technical Committee

T11 Chair
Steven L. Wilson
Brocade
1745 Technology Drive
San Jose, CA 95131
USA

Telephone: (408) 333-8128
Fax: (408) 392-6655
Email: swilson@brocade.com

T11 Vice-Chair
Claudio Desanti
Cisco Systems, Inc.
170 W. Tasman Dr.
San Jose, CA 95134
USA

Telephone:(408) 853-9172
Fax: (408) 853-9172
Email: cds@cisco.com

T11 Web Site: http://www.T11.org

INCITS Secretariat
Suite 610
1101 K St, NW
Washington, DC 20005
USA

Telephone: 202-737-8888
Web site: http://www.incits.org
Email: incits@itic.org

Information Technology Industry Council
Web site: http://www.itic.org

Document Distribution
INCITS Online Store
managed by:

Techstreet
3916 Ranchero Dr.
Ann Arbor, MI 48108
USA

Web site: http://www.techstreet.com/incitsgate.tmpl
Telephone: (800) 699-9277

Revision Information

Revision 0.82 (5 February 2013)

a) Added reference to incorportated comments from T11/13-023v0
b) Changed incorrect document number T11/12-229v4 to T11/12-226v4 in history for Revision 0.79

Revision 0.81 (30 January 2013)

c) Incorporated comments from T11/13-023v0

Revision 0.80 (3 December 2012)

d) Incorporated T11/12-448v0
e) Change bars indicate differences between Revision 0.77 and 0.80.

Revision 0.79 (28 November 2012)

f) Incorportated T11/12-226v4
g) Incorportated T11/12-297v1
h) Incorportated T11/12-337v2

Revision 0.77 (18 June 2010)

a) Incorporated T11/10-029v1 (“Remove legacy specifications”) amended as agreed in minutes of meeting 8 June 2010 (approved by work group 8 June 2010). Amendments are marked with change bars.
b) Removed APIs for obsolete ELSs RPL (Read Port List) and RPS (Read Port Status). Lines bracketing removals are marked with change bars.
c) Updated contacts and officers informationin front matter.
d) Updated ANSI references to current versions.

Revision 0.75 (13 May 2010)

a) Incorporated T11/09-470v0 (“What does Mandatory mean in SM-HBA-2?”) amended as agreed in minutes of meeting 8 October 2009 (approved by work group 8 October 2009).
b) Incorporated the changes defined in T11/10-099v1 (“ISO SM-HBA with backward compatibility amendment”), modified to account for their being defined for a different version of the HBA-API (directed by work group 30 March 2010).

Revision 0.70 (25 September 2009)

a) Incorporated new SAS states and speed as defined in T11/09-143v0, Proposals for SM-HBA-2 (approved by work group 6 August 2009).
b) Incorporated T11/09-399v0 amended as agreed in minutes of meeting 6 August 2009 (approved by work group 6 August 2009).

Revision 0.60 (30 July 2009)

a) Incorporated T11/09-142v0, Errata against several HBA API generations, as amended by T11/09-298v0, Comments on corrections (approved by work group 4 June 2009).
b) Changed the names of the SAS BROADCAST (RESERVED) events to match SAS-2 (approved by work group 4 June 2009).

Revision 0.50 (11 January 2008)

a) Incorporated T11/08-016v0, Virtualization Attributes Specification, (approved by work group 4 December 2007).

Revision 0.40 (18 October 2007)

a) Changed incorrect document number T11/07-060v1 to T11/07-040v1 in history for Revision 0.30.
b) Decimal separator changed per T11/07-410v1 (approved by work group 9 October 2007).
c) Incorporated T11/07-247v1, SMHBA_ScsiManagementIn/Out, (approved by work group 7 August 2007).

Revision 0.30 (20 March 2007)

a) Incorporated proposal T11/07-040v1 (approved by SM-API work group 7 February 2007)

Revision 0.20 (20 December 2006)

a) Incorporated proposal T11/06-714v1 (approved by SM-API work group 6 December 2006)

Revision 0.10 (28 September 2006)

a) Incorporated SM-HBA version 7.
b) Made changes requested by the ANSI editor for SM-HBA version 7.
c) Made minor editorial adjustments to front matter to match the practices of the ANSI editor.
d) Converted to a document template similar to the T10 template.
e) Updated several of the reference standards.
f) Redrew figures 1 and 2 using native FrameMaker drawing tools. Made some changes to bring their content up to date.
g) Made several (but not all) self-reference corrections, e.g.:
   A) Changed “SM-HBA” to “this standard”;
   B) Changed “SM-HBA compliant” to “compliant with this standard”; and
   C) Did not change “SM-HBA Functions” and similar titular usages.

Draft

American National Standard
for Information Technology

Storage Management - Host Bus Adapter 2nd Generation (SM-HBA-2)

Secretariat
Information Technology Industry Council

Approved mm.dd.yy

American National Standards Institute, Inc.

ABSTRACT

A standard Application Programming Interface defines a scope within which and a grammar by which it is possible to write application software without attention to vendor-specific infrastructure behavior. This standard defines a standard Application Programming Interface the scope of which is management of Fibre Channel and Serial Attached SCSI Host Bus Adapters and use of specific Fibre Channel and Serial Attached SCSI facilities for the discovery and management of the components of Fibre Channel Storage Area Network and Serial Attached SCSI domain.

This standard is to be used in conjunction with the Fibre Channel, Serial Attached SCSI and SCSI families of standards.

Draft

American National Standard

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer.

Consensus is established when, in the judgment of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that effort be made towards their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he has approved the standards or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give interpretation on any American National Standard. Moreover, no person shall have the right or authority to issue an interpretation of an American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the secretariat or sponsor whose name appears on the title page of this standard.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken periodically to reaffirm, revise, or withdraw this standard. Purchasers of American National Standards may receive current information on all standards by calling or writing the American National Standards Institute.

Caution: The developers of this standard have requested that holders of patents that may be required for the implementation of the standard disclose such patents to the publisher. However, neither the developers nor the publisher have undertaken a patent search in order to identify which, if any, patents may apply to this standard. As of the date of publication of this standard and following calls for the identification of patents that may be required for the implementation of the standard,

no such claims have been made. No further patent search is conducted by the developer or publisher in respect to any standard it processes.

- OR -

notice of one or more claims has been received. By publication of this standard, no position is taken with respect to the validity of this claim orof any rights in connection therewith. The known patent holder has, however, filed a statement of willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to pplicants desiring to obtain such a license. Details may be obtained from the publisher.

No further patent search is conducted by the developer or publisher in respect to any standard it processes. No representation is made or implied that licenses are not required to avoid infringement in the use of this standard.

Published by

American National Standards Institute
25 West 43rd Street, New York, NY 10036

Table of Contents

Topic **Page**

Revision Information - - - - - iii
Revision 0.82 (5 February 2013) - - - - - iii
Revision 0.81 (30 January 2013) - - - - - iii
Revision 0.80 (3 December 2012) - - - - - iii
Revision 0.79 (28 November 2012) - - - - - iii
Revision 0.77 (18 June 2010) - - - - - iii
Revision 0.75 (13 May 2010) - - - - - iii
Revision 0.70 (25 September 2009) - - - - - iii
Revision 0.60 (30 July 2009) - - - - - iii
Revision 0.50 (11 January 2008) - - - - - iii
Revision 0.40 (18 October 2007) - - - - - iv
Revision 0.30 (20 March 2007) - - - - - iv
Revision 0.20 (20 December 2006) - - - - - iv
Revision 0.10 (28 September 2006) - - - - - iv

Table of Contents - - - - - viii

List of Tables - - - - - xxii

List of Figures - - - - - xxiv

Foreword - - - - - xxv

Introduction - - - - - xxix

Acknowledgements - - - - - xxxi

1 Scope - - - - - 1

2 Normative References - - - - - 4
2.1 Normative references - - - - - 4
2.2 Approved references - - - - - 4
2.3 References under development - - - - - 5
2.4 IETF references - - - - - 5
2.5 Other references - - - - - 6

3 Definitions, symbols, abbreviations, and conventions - - - - - 7
3.1 Definitions - - - - - 7
3.2 Symbols and abbreviations - - - - - 12
3.3 Keywords - - - - - 13
3.4 Conventions - - - - - 13
3.5 Notation for Procedures and Functions - - - - - 14

4 General Constraints - - - - - 15
4.1 Software Structure - - - - - 15
4.2 Backwards Compatibility - - - - - 15
4.3 C language - - - - - 15
4.4 Operating System Dependencies - - - - - 16

5 Software Structure and Behavior - - - - - 17
5.1 Overview - - - - - 17
5.2 Software Structure - - - - - 17
5.2.1 OS specific structure - - - - - 17
5.2.2 OS independent structure - - - - - 17

5.3 Persistence of Identity ----- 18
5.4 HBA Configuration Rediscovery Effect on the API ----- 18
5.4.1 Introduction ----- 18
5.4.2 HBA_STATUS_ERROR_STALE_DATA ----- 19
5.4.3 Semistatic table model ----- 19
5.5 Multiuse considerations ----- 20

6 Attributes and Data Structures ----- 21
6.1 Basic Attribute Types ----- 21
6.2 Status Return Values ----- 22
6.3 Adapter Attributes ----- 24
6.3.1 Generic Adapter ----- 24
6.3.1.1 Generic Adapter Requirements ----- 24
6.3.1.2 Generic Adapter Attribute Data Declarations ----- 24
6.3.1.3 Generic Adapter Attribute Specifications ----- 25
6.3.1.3.1 HBAHandle ----- 25
6.3.1.3.2 HBAOptions ----- 25
6.3.1.3.3 Manufacturer ----- 25
6.3.1.3.4 SerialNumber ----- 25
6.3.1.3.5 Model ----- 26
6.3.1.3.6 ModelDescription ----- 26
6.3.1.3.7 HardwareVersion ----- 26
6.3.1.3.8 DriverVersion ----- 26
6.3.1.3.9 OptionROMVersion ----- 26
6.3.1.3.10 FirmwareVersion ----- 26
6.3.1.3.11 VendorSpecificID ----- 26
6.3.1.3.12 DriverName ----- 26
6.3.1.3.13 HBASymbolicName ----- 27
6.3.1.3.14 RedundantOptionROMVersion ----- 27
6.3.1.3.15 RedundantFirmwareVersion ----- 27
6.4 Adapter Bus Address ----- 27
6.4.1 Generic Adapter Bus Address ----- 27
6.4.1.1 Generic Adapter Bus Address Requirements ----- 27
6.4.1.2 Generic Adapter Bus Address Attribute Data Declarations ----- 27
6.4.1.3 Generic Adapter Bus Address Attribute Descriptions ----- 28
6.4.1.3.1 Type ----- 28
6.4.1.3.2 Address ----- 28
6.4.2 PCI Adapter Bus Address Attributes ----- 28
6.4.2.1 PCI Adapter Bus Address Requirements ----- 28
6.4.2.2 PCI Adapter Bus Address Attribute Data Declarations ----- 28
6.4.2.3 PCI Adapter Bus Address Attribute Specification ----- 28
6.4.2.3.1 BusNumber ----- 28
6.4.2.3.2 DeviceNumber ----- 28
6.4.2.3.3 FunctionNumber ----- 28
6.5 Port Attributes ----- 28
6.5.1 Generic Port ----- 28
6.5.1.1 Generic Port Requirements ----- 28
6.5.1.2 Generic Port Attribute Data Declarations ----- 29
6.5.1.3 Generic Port Attribute Specifications ----- 29
6.5.1.3.1 PortHandle ----- 29
6.5.1.3.2 PortType ----- 29
6.5.1.3.3 PortState ----- 30
6.5.1.3.4 OSDeviceName ----- 30
6.5.1.3.5 PortSpecificAttributes ----- 31

6.5.2 FC Port ---31
6.5.2.1 FC Port Requirements ---31
6.5.2.2 FC Port Attribute Data Declarations ---31
6.5.2.3 FC Port Attribute Specifications ---32
6.5.2.3.1 NodeWWN ---32
6.5.2.3.2 PortWWN ---32
6.5.2.3.3 AddressIdentifier ---32
6.5.2.3.4 PortSupportedClassofService ---32
6.5.2.3.5 PortSupportedFc4Types ---32
6.5.2.3.6 PortActiveFc4Types ---33
6.5.2.3.7 PortSymbolicName ---33
6.5.2.3.8 NumberofDiscoveredPorts ---33
6.5.3 SAS Port ---33
6.5.3.1 SAS Port Requirements ---33
6.5.3.2 SAS Port Attribute Data Declarations ---33
6.5.3.3 SAS Port Attribute Specifications ---34
6.5.3.3.1 PortProtocol ---34
6.5.3.3.2 LocalSASAddress ---34
6.5.3.3.3 AttachedSASAddress ---34
6.5.3.3.4 NumberofDiscoveredPorts ---34
6.6 Phy Attributes ---34
6.6.1 Generic Phy ---34
6.6.1.1 Generic Phy Requirements ---34
6.6.1.2 Generic Phy Attributes Data Declarations ---34
6.6.1.3 Generic Phy Attributes Specifications ---35
6.6.1.3.1 PhyHandle ---35
6.6.1.3.2 PhyType ---35
6.6.1.3.3 PhySpecificAttributes ---35
6.6.2 FC Phy ---35
6.6.2.1 FC Phy Requirements ---35
6.6.2.2 FC Phy Attribute Data Declaration ---35
6.6.2.3 FC Phy Attribute Specifications ---37
6.6.2.3.1 PhyOptions ---37
6.6.2.3.2 PhySupportedSpeed ---38
6.6.2.3.3 PhySpeed ---38
6.6.2.3.4 PhyState ---38
6.6.2.3.5 PhyProtocol ---38
6.6.2.3.6 MediaType ---38
6.6.2.3.7 MaxFrameSize ---38
6.6.3 SAS Phy ---38
6.6.3.1 SAS Phy requirements ---38
6.6.3.2 SAS Phy Attribute Data Declaration ---38
6.6.3.3 SAS Phy Attribute Specifications ---39
6.6.3.3.1 PhyIdentifier ---39
6.6.3.3.2 NegotiatedLinkRate ---39
6.6.3.3.3 ProgrammedMinLinkRate ---39
6.6.3.3.4 HardwareMinLinkRate ---39
6.6.3.3.5 ProgrammedMaxLinkRate ---39
6.6.3.3.6 HardwareMaxLinkRate ---39
6.6.3.3.7 domainPortWWN ---40
6.6.4 Ethernet Phy ---40
6.6.4.1 Ethernet Phy requirements ---40
6.6.4.2 Ethernet Phy Attributes Data Declarations ---40
6.6.4.3 Ethernet Phy Attributes Specifications ---41

6.6.4.3.1 PhyOptions ----41
6.6.4.3.2 PhySupportedSpeed ----41
6.6.4.3.3 PhySpeed ----41
6.6.4.3.4 PhyState ----41
6.6.4.3.5 MediaType ----41
6.6.4.3.6 MaxFrameSize ----41
6.6.4.3.7 VLANMask ----41
6.7 N_Port Controller ----42
6.7.1 Generic N_Port Controller ----42
6.7.1.1 Generic N_Port Controller Requirements ----42
6.7.1.2 Generic N_Port Controller Attributes Data Declarations ----42
6.7.1.3 N_Port Controller Attributes Specifications ----42
6.7.1.3.1 NPCHandle ----42
6.7.1.3.2 NPCType ----42
6.7.1.3.3 NPCSpecificAttributes ----43
6.7.2 FC N_Port Controller ----43
6.7.2.1 FC N_Port Controller requirements ----43
6.7.2.2 FC N_Port Controller Attributes Data Declarations ----43
6.7.2.3 FC N_Port Controller Attributes Specifications ----43
6.7.2.3.1 FCNOptions ----43
6.7.2.3.2 VFIDMask ----44
6.7.2.3.3 CoreNPortName ----44
6.7.2.3.4 CoreSwitchName ----44
6.7.2.3.5 PortVfid ----44
6.7.3 ENode FCoE Controller ----45
6.7.3.1 Overview ----45
6.7.3.2 ENode FCoE Controller requirements ----45
6.7.3.3 ENode FCoE Controller Attributes Data Declarations ----45
6.7.3.4 ENode FCoE Controller Attributes Specifications ----46
6.7.3.4.1 FCoECtlrOptions ----46
6.7.3.4.2 ENode MAC ----46
6.7.3.4.3 FCoECtlrVLTag ----46
6.7.3.4.4 FCFMACAddr ----46
6.7.3.4.5 MaxFCoESizeVerifiedBit ----46
6.7.3.4.6 AvailableForLoginBit ----46
6.7.4 FCoE Link Endpoint ----46
6.7.4.1 FCoE Link Endpoint requirements ----46
6.7.4.2 FCoE Link Endpoint Attributes Data Declarations ----46
6.7.4.3 ENode Link Endpoint Specifications ----47
6.7.4.3.1 LEPVNPortMAC ----47
6.7.4.3.2 LEPFCFMAC ----47
6.7.4.3.3 LEPVLTag ----47
6.7.4.3.4 BeaconPeriod ----47
6.7.4.3.5 FKAADVPeriod ----47
6.8 Fabric Attributes ----47
6.8.1 Fabric Info ----47
6.8.1.1 Fabric Info requirements ----47
6.8.1.2 Fabric Info Data Declarations ----47
6.8.1.3 Fabric Info Specifications ----48
6.8.1.3.1 FabricHandle ----48
6.8.1.3.2 FabricName ----48
6.8.1.3.3 Flags ----48
6.8.1.3.4 Ratov ----48
6.8.1.3.5 Edtov ----48

6.9 Statistics ----48
6.9.1 Protocol Statistics ----48
6.9.1.1 Protocol Statistics requirements ----48
6.9.1.2 Protocol Statistics Data Declarations ----49
6.9.1.3 Protocol Statistics Attribute Specifications ----49
6.9.1.3.1 SecondsSinceLastReset ----49
6.9.1.3.2 InputRequests ----49
6.9.1.3.3 OutputRequests ----49
6.9.1.3.4 ControlRequests ----49
6.9.1.3.5 InputMegabytes ----49
6.9.1.3.6 OutputMegabytes ----49
6.9.2 Port Statistics ----50
6.9.2.1 Port Statistics requirements ----50
6.9.2.2 Port Statistics Data Declarations ----50
6.9.2.3 Port Statistics Descriptions ----50
6.9.2.3.1 SecondsSinceLastReset ----50
6.9.2.3.2 TxFrames ----50
6.9.2.3.3 RxFrames ----50
6.9.2.3.4 TxWords ----50
6.9.2.3.5 RxWords ----50
6.9.3 Phy Statistics ----51
6.9.3.1 Phy Statistics Data Declaration ----51
6.9.4 SAS Phy Statistics ----51
6.9.4.1 SAS Phy Statistics requirements ----51
6.9.4.2 SAS Phy Statistics Data Declaration ----51
6.9.4.3 SAS Phy Statistics Attribute Specifications ----51
6.9.4.3.1 SecondsSinceLastReset ----51
6.9.4.3.2 TxFrames ----51
6.9.4.3.3 TxWords ----51
6.9.4.3.4 RxFrames ----52
6.9.4.3.5 RxWords ----52
6.9.4.3.6 InvalidDwordCount ----52
6.9.4.3.7 RunningDisparityErrorCount ----52
6.9.4.3.8 LossofDwordSynchronizationCount ----52
6.9.4.3.9 PhyResetProblemCount ----52
6.9.5 FC Phy Statistics ----52
6.9.5.1 FC Phy Statistics requirements ----52
6.9.5.2 FC Phy Statistics Data Declaration ----53
6.9.5.3 FC Phy Statistics Attribute Specifications ----53
6.9.5.3.1 SecondsSinceLastReset ----53
6.9.5.3.2 TxFrames ----53
6.9.5.3.3 RxFrames ----53
6.9.5.3.4 TxWords ----53
6.9.5.3.5 RxWords ----53
6.9.5.3.6 LIPCount ----53
6.9.5.3.7 NOSCount ----54
6.9.5.3.8 ErrorFrames ----54
6.9.5.3.9 DumpedFrames ----54
6.9.5.3.10 LinkFailureCount ----54
6.9.5.3.11 LossOfSyncCount ----54
6.9.5.3.12 LossOfSignalCount ----54
6.9.5.3.13 PrimitiveSeqProtocolErrCount ----54
6.9.5.3.14 InvalidTxWordCount ----54
6.9.5.3.15 Invalid CRC Count ----54

6.9.5.3.16 FLOGICount ----- 54
6.9.5.3.17 FLOGOCount ----- 55
6.9.6 Ethernet Phy Statistics ----- 55
6.9.6.1 Ethernet Phy Statistics Compliance ----- 55
6.9.6.2 Ethernet Phy Statistics Data Declaration ----- 55
6.9.6.3 Ethernet Phy Statistics Attribute Specifications ----- 55
6.9.6.3.1 SecondsSinceLastReset ----- 55
6.9.6.3.2 TxENFrames ----- 55
6.9.6.3.3 TxENBytes ----- 55
6.9.6.3.4 RxENFrames ----- 55
6.9.6.3.5 RxENBytes ----- 56
6.9.6.3.6 LinkFailureCount ----- 56
6.9.6.3.7 SymbolErrorCount ----- 56
6.9.6.3.8 ErroredBlockCount ----- 56
6.9.6.3.9 FCSErrorCount ----- 56
6.9.7 FIP Statistics ----- 56
6.9.7.1 FIP Statistics requirements ----- 56
6.9.7.2 FIP Statistics Data Declarations ----- 56
6.9.7.3 FIP Statistics Descriptions ----- 57
6.9.7.3.1 SecondsSinceLastReset ----- 57
6.9.7.3.2 FIPVLANNotifications ----- 57
6.9.7.3.3 FCFTimeoutCount ----- 57
6.9.7.3.4 BEACONTimeoutCount ----- 57
6.9.7.3.5 FIPMulticastAdvertReceivedCount ----- 57
6.9.7.3.6 KeepAliveSentCount ----- 57
6.9.7.3.7 ClearVirtualLinksReceivedCount ----- 57
6.10 Target Port Attributes ----- 57
6.10.1 Target Port Attribute Data Declaration ----- 57
6.10.1.1 SMHBA_BIND_CAPABILITY ----- 57
6.10.1.2 SMHBA_BIND_TYPE ----- 58
6.10.1.3 SMHBA_ScsiId ----- 58
6.10.1.4 SMHBA_LUID ----- 58
6.10.1.5 SMHBA_PORTLUN ----- 58
6.10.1.6 Composite types ----- 58
6.10.2 Target Mapping and Persistent Binding Attribute Specifications ----- 59
6.10.2.1 Overview ----- 59
6.10.2.2 SMHBA_BIND_CAPABILITY ----- 59
6.10.2.3 SMHBA_BIND_TYPE ----- 59
6.10.2.4 SMHBA_SCSIID ----- 59
6.10.2.5 SMHBA_LUID ----- 60
6.10.2.6 PortWWN ----- 60
6.10.2.7 domainPortWWN ----- 60
6.10.2.8 TargetLun ----- 61
6.10.2.9 OSDeviceName ----- 61
6.10.2.10 ScsiBusNumber ----- 65
6.10.2.11 ScsiTargetNumber ----- 66
6.10.2.12 ScsiOSLun ----- 66
6.10.3 Persistent Binding Capabilities ----- 66
6.10.3.1 Persistent Binding Capability: SMHBA_CAN_BIND_TO_WWPN ----- 66
6.10.3.2 Persistent Binding Capability: SMHBA_CAN_BIND_TO_LUID ----- 66
6.10.3.3 Persistent Binding Capability: SMHBA_CAN_BIND_ANY_LUNS ----- 66
6.10.3.4 Persistent Binding Capability: SMHBA_CAN_ BIND_AUTOMAP ----- 67
6.10.4 Persistent Binding Setting Types ----- 67
6.10.4.1 Persistent Binding Type: SMHBA_BIND_TO_WWPN ----- 67

6.10.4.2 Persistent Binding Type: SMHBA_BIND_TO_LUID ----67
6.11 Asynchronous Event Notification Attributes ----67
6.11.1 Asynchronous Event Data Declarations ----67
6.11.1.1 Callback Handle ----67
6.11.1.2 HBA Add Category Event Type ----67
6.11.1.3 HBA Category Event Types ----68
6.11.1.4 Port Category Event Types ----68
6.11.1.5 Port Statistics Category Event Types ----68
6.11.1.6 Phy Statistics Category Event Types ----68
6.11.1.7 Target Category Event Types ----68
6.11.1.8 Link Category Event Types ----68
6.11.2 Asynchronous Event Attribute Specifications ----69
6.11.2.1 EventType ----69
6.12 Library Attributes ----70
6.12.1 Library Attribute Data Declarations ----70
6.12.2 Function Prototypes ----71
6.12.3 SM-HBA-2 Entry Point Data Declarations ----73
6.12.4 Entry Point Specifications ----75
6.12.5 Library Attribute Data Declarations ----75
6.12.6 Library Attribute Specifications ----75
6.12.6.1 Compliance ----75
6.12.6.2 LibPath ----75
6.12.6.3 VName ----75
6.12.6.4 VVersion ----75
6.12.6.5 build_date ----75
6.13 FC-3 Management Attributes ----76
6.13.1 FC-3 Management Data Declarations ----76
6.13.2 FC-3 Management Attribute Overview ----76
6.13.3 FC-3 Management Attribute Specifications ----76
6.13.3.1 FC-3 Management Requirements ----76
6.13.3.2 WWN ----76
6.13.3.3 unittype ----76
6.13.3.4 PortId ----76
6.13.3.5 NumberOfAttachedNodes ----77
6.13.3.6 IPVersion ----77
6.13.3.7 UDPPort ----77
6.13.3.8 IPAddress ----77
6.13.3.9 TopologyDiscoveryFlags ----77

7 Function Calls ----79
7.1 Overview ----79
7.2 Library Control Functions ----82
7.2.1 SMHBA2_GetVersion ----82
7.2.1.1 Format ----82
7.2.1.2 Description ----82
7.2.1.3 Arguments ----82
7.2.1.4 Return Values ----82
7.2.2 HBA_LoadLibrary ----82
7.2.2.1 Format ----82
7.2.2.2 Description ----82
7.2.2.3 Arguments ----83
7.2.2.4 Return Values ----83
7.2.3 HBA_FreeLibrary ----83
7.2.3.1 Format ----83

7.2.3.2 Description ----- 83
7.2.3.3 Arguments ----- 83
7.2.3.4 Return Values ----- 84
7.2.4 SMHBA2_RegisterLibrary ----- 84
7.2.4.1 Format ----- 84
7.2.4.2 Description ----- 84
7.2.4.3 Arguments ----- 84
7.2.4.4 Return Values ----- 84
7.2.5 SMHBA_GetWrapperLibraryAttributes ----- 85
7.2.5.1 Format ----- 85
7.2.5.2 Description ----- 85
7.2.5.3 Arguments ----- 85
7.2.5.4 Return Values ----- 85
7.2.6 SMHBA_GetVendorLibraryAttributes ----- 85
7.2.6.1 Format ----- 85
7.2.6.2 Description ----- 85
7.2.6.3 Arguments ----- 86
7.2.6.4 Return Values ----- 86
7.2.7 HBA_GetNumberOfAdapters ----- 86
7.2.7.1 Format ----- 86
7.2.7.2 Description ----- 86
7.2.7.3 Arguments ----- 86
7.2.7.4 Return Values ----- 87
7.3 Object Attribute Functions ----- 87
7.3.1 SMHBA2_GetAdapterHandleByIndex ----- 87
7.3.1.1 Format ----- 87
7.3.1.2 Description ----- 87
7.3.1.3 Arguments ----- 87
7.3.1.4 Return Values ----- 87
7.3.2 SMHBA2_GetAdapterAttributes ----- 88
7.3.2.1 Format ----- 88
7.3.2.2 Description ----- 88
7.3.2.3 Arguments ----- 88
7.3.2.4 Return Values ----- 88
7.3.3 SMHBA_GetNumberofPorts ----- 88
7.3.3.1 Format ----- 88
7.3.3.2 Description ----- 88
7.3.3.3 Arguments ----- 89
7.3.3.4 Return Values ----- 89
7.3.4 SMHBA2_GetAdapterBusAttributes ----- 89
7.3.4.1 Format ----- 89
7.3.4.2 Description ----- 89
7.3.4.3 Arguments ----- 89
7.3.4.4 Return Values ----- 90
7.3.5 SMHBA2_GetPortType ----- 90
7.3.5.1 Format ----- 90
7.3.5.2 Description ----- 90
7.3.5.3 Arguments ----- 90
7.3.5.4 Return Values ----- 91
7.3.6 SMHBA2_GetPortAttributes ----- 91
7.3.6.1 Format ----- 91
7.3.6.2 Description ----- 91
7.3.6.3 Arguments ----- 91
7.3.6.4 Return Values ----- 92

7.3.7 SMHBA2_GetPortAttributesByWWN ----92
7.3.7.1 Format ----92
7.3.7.2 Description ----92
7.3.7.3 Arguments ----92
7.3.7.4 Return Values ----93
7.3.8 SMHBA2_GetPhyType ----93
7.3.8.1 Format ----93
7.3.8.2 Description ----93
7.3.8.3 Arguments ----93
7.3.8.4 Return Values ----94
7.3.9 SMHBA2_GetPhyAttributes ----94
7.3.9.1 Format ----94
7.3.9.2 Description ----94
7.3.9.3 Arguments ----94
7.3.9.4 Return Values ----95
7.3.10 SMHBA2_GetPhyCtlrAttributes ----95
7.3.10.1 Format ----95
7.3.10.2 Description ----95
7.3.10.3 Arguments ----95
7.3.10.4 Return Values ----96
7.3.11 SMHBA2_GetFabricInfo ----96
7.3.11.1 Format ----96
7.3.11.2 Description ----96
7.3.11.3 Arguments ----96
7.3.11.4 Return Values ----97
7.4 Object Relationship functions ----97
7.4.1 SMHBA2_GetPortsOnAdapter ----97
7.4.1.1 Format ----97
7.4.1.2 Description ----97
7.4.1.3 Arguments ----97
7.4.1.4 Return Values ----98
7.4.2 SMHBA2_GetAdapterForPort ----98
7.4.2.1 Format ----98
7.4.2.2 Description ----98
7.4.2.3 Arguments ----98
7.4.2.4 Return Values ----99
7.4.3 SMHBA2_GetLEPForPort ----99
7.4.3.1 Format ----99
7.4.3.2 Description ----99
7.4.3.3 Arguments ----99
7.4.3.4 Return Values ----100
7.4.4 SMHBA2_GetDiscoveredPorts ----100
7.4.4.1 Format ----100
7.4.4.2 Description ----100
7.4.4.3 Arguments ----100
7.4.4.4 Return Values ----101
7.4.5 SMHBA2_GetPhysOnAdapter ----101
7.4.5.1 Format ----101
7.4.5.2 Description ----101
7.4.5.3 Arguments ----101
7.4.5.4 Return Values ----102
7.4.6 SMHBA2_GetAdapterForPhy ----102
7.4.6.1 Format ----102
7.4.6.2 Description ----102

7.4.6.3 Arguments ----102
7.4.6.4 Return Values ----103
7.4.7 SMHBA2_GetPortsOnPhy ----103
7.4.7.1 Format ----103
7.4.7.2 Description ----103
7.4.7.3 Arguments ----103
7.4.7.4 Return Values ----104
7.4.8 SMHBA2_GetPhysForPort ----104
7.4.8.1 Format ----104
7.4.8.2 Description ----104
7.4.8.3 Arguments ----104
7.4.8.4 Return Values ----105
7.4.9 SMHBA2_GetCtlrForPhy ----105
7.4.9.1 Format ----105
7.4.9.2 Description ----105
7.4.9.3 Arguments ----105
7.4.9.4 Return Values ----106
7.4.10 SMHBA2_GetPhyForCtlr ----106
7.4.10.1 Format ----106
7.4.10.2 Description ----106
7.4.10.3 Arguments ----106
7.4.10.4 Return Values ----107
7.4.11 SMHBA2_GetFabricsForCtlr ----107
7.4.11.1 Format ----107
7.4.11.2 Description ----107
7.4.11.3 Arguments ----107
7.4.11.4 Return Values ----108
7.4.12 SMHBA2_GetCtlrsForFabrics ----108
7.4.12.1 Format ----108
7.4.12.2 Description ----108
7.4.12.3 Arguments ----108
7.4.12.4 Return Values ----109
7.4.13 SMHBA2_GetFabricForPort ----109
7.4.13.1 Format ----109
7.4.13.2 Description ----109
7.4.13.3 Arguments ----109
7.4.13.4 Return Values ----110
7.4.14 SMHBA2_GetPortsForFabric ----110
7.4.14.1 Format ----110
7.4.14.2 Description ----110
7.4.14.3 Arguments ----110
7.4.14.4 Return Values ----111
7.5 Statistics Functions ----111
7.5.1 SMHBA2_GetPortStatistics ----111
7.5.1.1 Format ----111
7.5.1.2 Description ----111
7.5.1.3 Arguments ----111
7.5.1.4 Return Values ----112
7.5.2 SMHBA2_GetProtocolStatistics ----112
7.5.2.1 Format ----112
7.5.2.2 Description ----112
7.5.2.3 Arguments ----112
7.5.2.4 Return Values ----113
7.5.3 SMHBA2_GetPhyStatistics ----113

7.5.3.1 Format ----113
7.5.3.2 Description ----113
7.5.3.3 Arguments ----113
7.5.3.4 Return Values ----114
7.5.4 SMHBA2_GetFIPStatistics ----114
7.5.4.1 Format ----114
7.5.4.2 Description ----114
7.5.4.3 Arguments ----114
7.5.4.4 Return Values ----115
7.6 Fabric and Domain Management Functions ----115
7.6.1 HBA_SendCTPassThruV2 ----115
7.6.1.1 Format ----115
7.6.1.2 Description ----115
7.6.1.3 Arguments ----115
7.6.1.4 Return Values ----116
7.6.2 HBA_SetRNIDMgmtInfo ----116
7.6.2.1 Format ----116
7.6.2.2 Description ----116
7.6.2.3 Arguments ----116
7.6.2.4 Return Values ----117
7.6.3 HBA_GetRNIDMgmtInfo ----117
7.6.3.1 Format ----117
7.6.3.2 Description ----117
7.6.3.3 Arguments ----117
7.6.3.4 Return Values ----118
7.6.4 HBA_SendRNIDV2 ----118
7.6.4.1 Format ----118
7.6.4.2 Description ----118
7.6.4.3 Arguments ----119
7.6.4.4 Return Values ----119
7.6.5 HBA_SendSRL ----120
7.6.5.1 Format ----120
7.6.5.2 Description ----120
7.6.5.3 Arguments ----120
7.6.5.4 Return Values ----121
7.6.6 HBA_SendLIRR ----121
7.6.6.1 Format ----121
7.6.6.2 Description ----121
7.6.6.3 Arguments ----122
7.6.6.4 Return Values ----122
7.6.7 HBA_SendRLS ----123
7.6.7.1 Format ----123
7.6.7.2 Description ----123
7.6.7.3 Arguments ----123
7.6.7.4 Return Values ----124
7.6.8 SMHBA_SendTEST ----124
7.6.8.1 Format ----124
7.6.8.2 Description ----124
7.6.8.3 Arguments ----125
7.6.8.4 Return Values ----125
7.6.9 SMHBA_SendECHO ----126
7.6.9.1 Format ----126
7.6.9.2 Description ----126
7.6.9.3 Arguments ----126

7.6.9.4 Return Values ----- 127
7.6.10 SMHBA_SendSMPPassThru ----- 127
7.6.10.1 Format ----- 127
7.6.10.2 Description ----- 128
7.6.10.3 Return Values ----- 128
7.7 SM-HBA Target Information Functions ----- 129
7.7.1 SMHBA_GetBindingCapability ----- 129
7.7.1.1 Format ----- 129
7.7.1.2 Description ----- 129
7.7.1.3 Arguments ----- 130
7.7.1.4 Return Values ----- 130
7.7.2 SMHBA_GetBindingSupport ----- 131
7.7.2.1 Format ----- 131
7.7.2.2 Description ----- 131
7.7.2.3 Arguments ----- 131
7.7.2.4 Return Values ----- 132
7.7.3 SMHBA_SetBindingSupport ----- 132
7.7.3.1 Format ----- 132
7.7.3.2 Description ----- 132
7.7.3.3 Arguments ----- 133
7.7.3.4 Return Values ----- 133
7.7.4 SMHBA_GetTargetMapping ----- 134
7.7.4.1 Format ----- 134
7.7.4.2 Description ----- 134
7.7.4.3 Arguments ----- 135
7.7.4.4 Return Values ----- 135
7.7.5 SMHBA_GetPersistentBinding ----- 136
7.7.5.1 Format ----- 136
7.7.5.2 Description ----- 136
7.7.5.3 Arguments ----- 136
7.7.5.4 Return Values ----- 137
7.7.6 SMHBA_SetPersistentBinding ----- 138
7.7.6.1 Format ----- 138
7.7.6.2 Description ----- 138
7.7.6.3 Arguments ----- 138
7.7.6.4 Return Values ----- 139
7.7.7 SMHBA_RemovePersistentBinding ----- 139
7.7.7.1 Format ----- 139
7.7.7.2 Description ----- 140
7.7.7.3 Arguments ----- 140
7.7.7.4 Return Values ----- 141
7.7.8 SMHBA_RemoveAllPersistentBindings ----- 141
7.7.8.1 Format ----- 141
7.7.8.2 Description ----- 141
7.7.8.3 Arguments ----- 142
7.7.8.4 Return Values ----- 142
7.7.9 SMHBA_GetLUNStatistics ----- 143
7.7.9.1 Format ----- 143
7.7.9.2 Description ----- 143
7.7.9.3 Arguments ----- 143
7.7.9.4 Return Values ----- 143
7.8 SCSI Information Functions ----- 144
7.8.1 SMHBA_ScsiInquiry ----- 144
7.8.1.1 Format ----- 144

7.8.1.2 Description ----144
7.8.1.3 Arguments ----144
7.8.1.4 Return Values ----146
7.8.2 SMHBA_ScsiReportLuns ----147
7.8.2.1 Format ----147
7.8.2.2 Description ----147
7.8.2.3 Arguments ----147
7.8.2.4 Return Values ----148
7.8.3 SMHBA_ScsiReadCapacity ----149
7.8.3.1 Format ----149
7.8.3.2 Description ----149
7.8.3.3 Arguments ----149
7.8.3.4 Return Values ----150
7.8.4 SMHBA_ScsiManagementIn ----151
7.8.4.1 Format ----151
7.8.4.2 Description ----151
7.8.4.3 Arguments ----151
7.8.4.4 Return Values ----153
7.8.5 SMHBA_ScsiManagementOut ----154
7.8.5.1 Format ----154
7.8.5.2 Description ----154
7.8.5.3 Arguments ----154
7.8.5.4 Return Values ----156
7.9 Event Handling Functions ----157
7.9.1 Overview of SM-HBA Event Reporting ----157
7.9.1.1 Asynchronous Event Reporting Behavior Model ----157
7.9.1.2 Registration for Events with diverse HBA specific software ----158
7.9.2 SMHBA_RegisterForAdapterAddEvents ----158
7.9.2.1 Format ----158
7.9.2.2 Description ----158
7.9.2.3 Arguments ----159
7.9.2.4 Return Values ----159
7.9.2.5 Callback Arguments ----159
7.9.3 SMHBA_RegisterForAdapterEvents ----159
7.9.3.1 Format ----159
7.9.3.2 Description ----160
7.9.3.3 Arguments ----160
7.9.3.4 Return Values ----160
7.9.3.5 Callback Arguments ----160
7.9.4 SMHBA_RegisterForAdapterPortEvents ----160
7.9.4.1 Format ----160
7.9.4.2 Description ----161
7.9.4.3 Arguments ----161
7.9.4.4 Return Values ----162
7.9.4.5 Callback Arguments ----162
7.9.5 SMHBA_RegisterForAdapterPortStatEvents ----162
7.9.5.1 Format ----162
7.9.5.2 Description ----163
7.9.5.3 Arguments ----163
7.9.5.4 Return Values ----164
7.9.5.5 Callback Arguments ----164
7.9.6 SMHBA2_RegisterForAdapterPhyStatEvents ----164
7.9.6.1 Format ----164
7.9.6.2 Description ----165

7.9.6.3 Arguments ----165
7.9.6.4 Return Values ----166
7.9.6.5 Callback Arguments ----166
7.9.7 SMHBA_RegisterForTargetEvents ----166
7.9.7.1 Format ----166
7.9.7.2 Description ----167
7.9.7.3 Arguments ----167
7.9.7.4 Return Values ----168
7.9.7.5 Callback Arguments ----168
7.9.8 HBA_RegisterForLinkEvents ----169
7.9.8.1 Format ----169
7.9.8.2 Description ----169
7.9.8.3 Arguments ----169
7.9.8.4 Return Values ----170
7.9.8.5 Callback Arguments ----170
7.9.9 HBA_RemoveCallback ----170
7.9.9.1 Format ----170
7.9.9.2 Description ----170
7.9.9.3 Arguments ----171
7.9.9.4 Return Values ----171

8 Configuration ----172
8.1 Overview ----172
8.2 Win32 ----172
8.3 Unix ----173

Annex A: SM-HBA-2 Compliance Requirements ----174
A.1 Overview ----174
A.2 Functions ----175
A.3 Generic Adapter Attributes ----178
A.4 Generic Bus Attributes ----179
A.5 PCI Bus Attributes ----180
A.6 Generic Port Attributes ----180
A.7 FC_Port Attributes ----181
A.8 SAS Port Attributes ----182
A.9 Generic Phy Attributes ----182
A.10 FC Phy Attributes ----183
A.11 SAS Phy Attribute ----184
A.12 Enet Phy Attributes ----185
A.13 Generic N_Port Controller Attributes ----186
A.14 FC N_Port Controller Attributes ----186
A.15 ENode FCoE Controller Attributes ----187
A.16 FCoE Link Endpoint ----188
A.17 Fabric Info ----188
A.18 Protocol Statistics ----189
A.19 Port Statistics ----189
A.20 SAS Phy Statistics ----190
A.21 FC Phy Statistics ----191
A.22 Enet Phy Statistics ----192
A.23 FIP Statistics ----192
A.24 FC-3 Management Attributes ----193
A.25 SM-HBA-2 Library Attributes ----194

Annex B: Bibliography ----196

List of Tables

Table	Page
1 Preferred format for SMHBA2_PortAttributes OSDeviceName	30
2 Type of attributes structure pointer in PhyAttributes	35
3 Type of attributes structure pointer in NPCAttributes	43
4 Preferred format for logical unit OSDeviceName	62
5 Asynchronous event type codes	69
6 Function Summary and Requirements	79
7 Returned Function Values for SMHBA2_GetVersion	82
8 Returned Function Values for HBA_LoadLibrary	83
9 Returned Function Values for HBA_FreeLibrary	84
10 Returned Function Values for SMHBA2_RegisterLibrary	84
11 Returned Function Values for SMHBA_GetWrapperLibraryAttributes	85
12 Returned Function Values for SMHBA_GetVendorLibraryAttributes	86
13 Returned Function Values for SMHBA2_GetAdapterHandleByIndex	87
14 Returned Function Values for SMHBA2_GetAdapterAttributes	88
15 Returned Function Values for SMHBA_GetNumberofPorts	89
16 Returned Function Values for SMHBA2_GetAdapterBusAttributes	90
17 Returned Function Values for SMHBA2_GetPortType	91
18 Returned Function Values for SMHBA2_GetPortAttributes	92
19 Returned Function Values for SMHBA2_GetPortAttributesByWWN	93
20 Returned Function Values for SMHBA2_GetPhyType	94
21 Returned Function Values for SMHBA2_GetPhyAttributes	95
22 Returned Function Values for SMHBA2_GetPhyCtlrAttributes	96
23 Returned Function Values for SMHBA2_GetFabricInfo	97
24 Returned Function Values for SMHBA2_GetPortsOnAdapter	98
25 Returned Function Values for SMHBA2_GetAdapterForPort	99
26 Returned Function Values for SMHBA2_GetLEPForPort	100
27 Returned Function Values for SMHBA2_GetDiscoveredPorts	101
28 Returned Function Values for SMHBA2_GetPhysOnAdapter	102
29 Returned Function Values for SMHBA2_GetAdapterForPhy	103
30 Returned Function Values for SMHBA2_GetPortsOnPhy	104
31 Returned Function Values for SMHBA2_GetPhysForPort	105
32 Returned Function Values for SMHBA2_GetCtlrForPhy	106
33 Returned Function Values for SMHBA2_GetPhyForCtlr	107
34 Returned Function Values for SMHBA2_GetFabricsForCtlr	108
35 Returned Function Values for SMHBA2_GetCtlrsForFabric	109
36 Returned Function Values for SMHBA2_GetFabricForPort	110
37 Returned Function Values for SMHBA2_GetPortsForFabric	111
38 Returned Function Values for SMHBA2_GetPortStatistics	112
39 Returned Function Values for SMHBA2_GetProtocolStatistics	113
40 Returned Function Values for SMHBA2_GetPhyStatistics	114
41 Returned Function Values for SMHBA2_GetFIPStatistics	115
42 Returned Function Values for HBA_SendCTPassThruV2	116
43 Returned Function Values for HBA_SetRNIDMgmtInfo	117
44 Returned Function Values for HBA_GetRNIDMgmtInfo	118
45 Returned Function Values for HBA_SendRNIDV2	119
46 Returned Function Values for HBA_SendSRL	121
47 Returned Function Values for HBA_SendLIRR	122
48 Returned Function Values for HBA_SendRLS	124
49 Returned Function Values for SMHBA_SendTEST	125
50 Returned Function Values for SMHBA_SendEcho	127

51 Returned Function Values for SMHBA_SendSMPPassThru ----------128
52 Returned Function Values for SMHBA_GetBindingCapability ----------130
53 Returned Function Values for SMHBA_GetBindingSupport ----------132
54 Returned Function Values for SMHBA_SetBindingSupport ----------133
55 Returned Function Values for SMHBA_GetTargetMapping ----------135
56 Returned Function Values for SMHBA_GetPersistentBinding ----------137
57 Returned Function Values for SMHBA_SetPersistentBinding ----------139
58 Returned Function Values for SMHBA_RemovePersistentBinding ----------141
59 Returned Function Values for SMHBA_RemoveAllPersistentBindings ----------142
60 Returned Function Values for SMHBA_GetLUNStatistics ----------143
61 Values for CDB_Byte1 ----------145
62 Returned Function Values for SMHBA_ScsiInquiry ----------146
63 Returned Function Values for SMHBA_ScsiReportLuns ----------148
64 Returned Function Values for SMHBA_ScsiReadCapacity ----------150
65 Returned Function Values for SMHBA_ScsiManagementIn ----------153
66 Returned Function Values for SMHBA_ScsiManagementOut ----------156
67 Returned Function Values for SMHBA_RegisterForAdapterAddEvents ----------159
68 Returned Function Values for SMHBA_RegisterForAdapterEvents ----------160
69 Returned Function Values for SMHBA_RegisterForAdapterPortEvents ----------162
70 Returned Function Values for SMHBA_RegisterForAdapterPortStatEvents ----------164
71 Returned Function Values for SMHBA2_RegisterForAdapterPhyStatEvents ----------166
72 Returned Function Values for SMHBA_RegisterForTargetEvents ----------168
73 Returned Function Values for HBA_RegisterForLinkEvents ----------170
74 Returned Function Values for HBA_RemoveCallback ----------171
A.1 General Function Requirements ----------175
A.2 Generic Adapter attributes ----------179
A.3 Generic Bus Attriutes ----------180
A.4 PCI Bus Attributes ----------180
A.5 Generic Port Attributes ----------181
A.6 FC_Port Attributes ----------181
A.7 SAS Port Attributes ----------182
A.8 Generic Phy Attributes ----------183
A.9 FC Phy Attributes ----------183
A.10 SAS Phy Attributes ----------184
A.11 Enet Phy Attributes ----------185
A.12 Generic N_Port Controller Attributes ----------186
A.13 FC N_Port Controller Attributes ----------186
A.14 ENode FCoE Controller Attributes ----------187
A.15 FCoE Link Endpoint ----------188
A.16 Fabric Info ----------188
A.17 Protocol Statistics ----------189
A.18 Port Statistics ----------189
A.19 SAS Phy Statistics ----------190
A.20 FC Phy Statistics ----------191
A.21 Enet Phy Statistics ----------192
A.22 FIP Statistics ----------193
A.23 FC-3 Management Attributes ----------194
A.24 Library Attributes ----------195

List of Figures

Figure **Page**

1 Context for this standard - 3
2 SM-HBA-2 system architecture -16

Foreword

(This foreword is not part of American National Standard INCITS.***:200x.)

Technical Committee T11 of Accredited Standards Committee INCITS developed this standard during 2003-2006. The standards approval process started in 2005. This document includes 11 annexes that are informative and are not considered part of the standard.

An HBA is hardware, typically on a host system and sometimes embedded on a RAID controller, that interfaces the host to storage subsystems or storage devices through an I/O bus. The I/O bus can be implemented through multiple interconnect technologies such as Fiber Channel (FC), Serial Attached SCSI (SAS), parallel SCSI etc. The storage subsystem, depending upon the interconnect technology, may be referred to as a "FC Fabric", a "SAS Domain", or a "bus".

An HBA could support multiple protocol mappings such as SSP - SCSI Serial Protocol (SCSI protocol mapping over SAS), STP - Serial ATA Transport Protocol (SATA protocol mapping over SAS), FCP - Fibre Channel Protocol (SCSI mapping over FC) etc. In addition, an HBA could simultaneously support multiple interconnect technologies such as SAS and FC.

In order to manage the HBA, the domain or fabric and the storage devices present in the environment, upper level management software applications requires information from HBAs in a consistent manner across operating systems, vendors, and platforms. The availability of such a consistent interface defined by the Storage Management - HBA API (see SM-HBA) and Fiber Channel - Switch API (see FC-SWAPI) have significantly reduced the interoperability issues associated with deploying a FC based Storage Area Network.

This standard specifies a standard API the scope of which is management of FC and SAS HBAs, and use of FC and SAS capabilities for discovery and management of the components of the respective fabric or domain. This proposed standard defines interfaces to the following capabilities:

a) observation and modification of descriptive and operational characteristics of physical and virtual HBAs and end ports;
b) management of security functions provided by an HBA and/or its end ports;
c) access to Fibre Channel Fabric Services;
d) access to Fibre Channel Extended Link Services necessary to comply with the FC-DA-2 manageability profile for Host Bus Adapters;
e) discovery and characterization of SCSI storage resources;
f) observation of HBA, end port, and storage access traffic statistics;
g) observation and modification of the availability and representation of SCSI storage resources to Operating System applications; and
h) timely and selective reporting of HBA and fabric configuration, status, and statistical events.

Requests for interpretation, suggestions for improvement and addenda, or defect reports are welcome. They should be sent to the INCITS Secretariat, InterNational Committee for Information Technology Standards, Information Technology Institute, 1250 Eye Street, NW, Suite 200, Washington, DC 20005-3922.

Users of this standard are encouraged to determine if there are standards in development or new versions of this standard that may extend or clarify technical information contained in this standard.

This standard was processed and approved for submittal to ANSI by the InterNational Committee for Information Technology Standards (INCITS). Committee approval of the standard does not necessarily imply that all committee members voted for approval. At the time of it approved this standard, INCITS had the following members:

Company	Representative

Editor's Note 1: <<Insert INCITS member list>>

Technical Committee T11 on Lower Level Interfaces, which reviewed this standard, had the following members:

Robert Snively, Chair
Claudio DeSanti, Vice-Chair
William R. Martin, Secretary

Company	Representative

Task Group T11.5 on Storage Management Interfaces, which developed and reviewed this standard, had the following members:

Roger Cummings, Chair
Scott Kipp, Vice-Chair
Ralph Weber, Secretary

Company	**Representative**

Introduction

This standard is divided into these clauses and annexes:

Clause 1 defines the scope of this standard and places it in context of other standards and standards projects.
Clause 2 enumerates the normative references that apply to this standard.
Clause 3 specifies definitions, symbols, and abbreviations.
Clause 4 presents general constraints on the design of this standard and on compliant implementations.
Clause 5 specifies constraints imposed on the structure and general behavior of implementations.
Clause 6 specifies data structures and attribute semantics.
Clause 7 specifies function calls.
Clause 8 specifies methods of configuring implementations.
Annex A is a normative specification identifying required and optional features.
Annex B is a bibliography of documents that are informatively referenced by this standard.

Acknowledgements

The Technical Editor would like to thank ....

1 Scope

A standard application programming interface (API) defines a scope within which, and a grammar by which it is possible to write application software without attention to vendor-specific infrastructure behavior. This standard specifies a standard API the scope of which is management of Fibre Channel (FC) and Serial Access SCSI (SAS) HBAs, and the use of FC and SAS capabilities for discovery and management of the components of the respective fabric or domain. This proposed standard defines interfaces to the following capabilities:

a) Monitoring and Control of Attributes and Capabilities of HBAs and End Ports.
b) Monitoring of HBA, End port, and Storage access traffic statistics.
c) Timely and selective reporting of HBA and fabric or domain configuration, status, and statistical events.
d) Access to Fibre Channel Fabric Services (see FC-FS-3).
e) Access to the Fibre Channel Extended Link Services necessary to comply with the manageability profile for HBAs recommended in FC-DA-2 (see FC-DA-2).
f) Access to SAS Management Protocol (SMP) Services for Expander Management (see SPL).
g) Discovery and Enumeration of storage resources availablue using FCP-4 (see FCP-4), SSP (see SPL), and STP (see SPL).

The primary goal of this standard is to provide a successor to SM-HBA that adds support for new Fibre Channel architecture features, including

a) Fibre Channel Virtual Fabrics;
b) Fibre Channel N_Port Virtualization;
c) virtual I/O interfaces to HBAs (e.g., PCI I/O Virtualization, see SRIOV 1.1); and
d) Fibre Channel over Ethernet (see FC-BB-6).

This standard is derived from SM-HBA (see SM-HBA), carrying forward many SM-HBA functions unchanged, while removing obsolete features and replacing functions that are inconsistent with new Fibre Channel architecture features. Although this standard does not obsolete SM-HBA as a reference for existing implementations, it is intended to replace SM-HBA as the recommended basis for new implementations.

Features of SM-HBA omitted from this standard include:

a) reporting of support for Hunt Groups, Multicast, and Class 1 Service;
b) N_Port characteristics specific to Single-Byte Command Codes;
c) mapping FC-HBA to InfiniBand™; and
d) polled event notification.

Functions of SM-HBA removed or replaced in this standard include:

a) HBA_RegisterLibrary;
b) HBA_RegisterLibraryV2;
c) HBA_GetWrapperLibraryAttributes;
d) HBA_GetVendorLibraryAttributes;
e) HBA_ResetStatistics;
f) HBA_GetAdapterName;
g) HBA_OpenAdapter;
h) HBA_OpenAdapterByWWN;
i) HBA_CloseAdapter;
j) HBA_RefreshInformation;
k) HBA_RefreshAdapterConfiguration;
l) HBA_GetAdapterAttributes;
m) HBA_GetAdapterPortAttributes;
n) HBA_GetDiscoveredPortAttributes;
o) HBA_GetPortAttributesByWWN;

p) HBA_GetPortStatistics;
q) HBA_GetFC4Statistics;
r) HBA_GetBindingCapability;
s) HBA_GetBindingSupport;
t) HBA_Support;
u) HBA_GetFcpTargetMapping;
v) HBA_GetFcpTargetMappingV2;
w) HBA_GetFcpPersistentBinding;
x) HBA_GetPersistentBindingV2;
y) HBA_SetPersistentBindingV2;
z) HBA_RemovePersistentBinding;
aa) HBA_RemoveAllPersistentBindings;
ab) HBA_GetFCPStatistics;
ac) HBA_SendScsiInquiry;
ad) HBA_ScsiInquiryV2;
ae) HBA_SendReportLUNs;
af) HBA_ScsiReportLunsV2;
ag) HBA_SendReadCapacity;
ah) HBA_ScsiReadCapacityV2;
ai) HBA_GetSBTargetMapping;
aj) HBA_GetSBStatistics;
ak) HBA_SBDskGetCapacity;
al) HBA_SendRPL;
am) HBA_SendRPS;
an) HBA_SendCTPassThru;
ao) HBA_SendRNID;
ap) HBA_GetEventBuffer
aq) HBA_RegisterForAdapterAddEvents
ar) HBA_RegisterForAdapterEvents
as) HBA_RegisterForAdapterPortEvents
at) HBA_RegisterForAdapterPortStatEvents
au) HBA_RegisterForTargetEvents.

This standard is to be used in conjunction with the Fibre Channel, Serial Attached SCSI and SCSI families of standards. At the time this standard was written its relationship to those standards was as shown in figure 1.

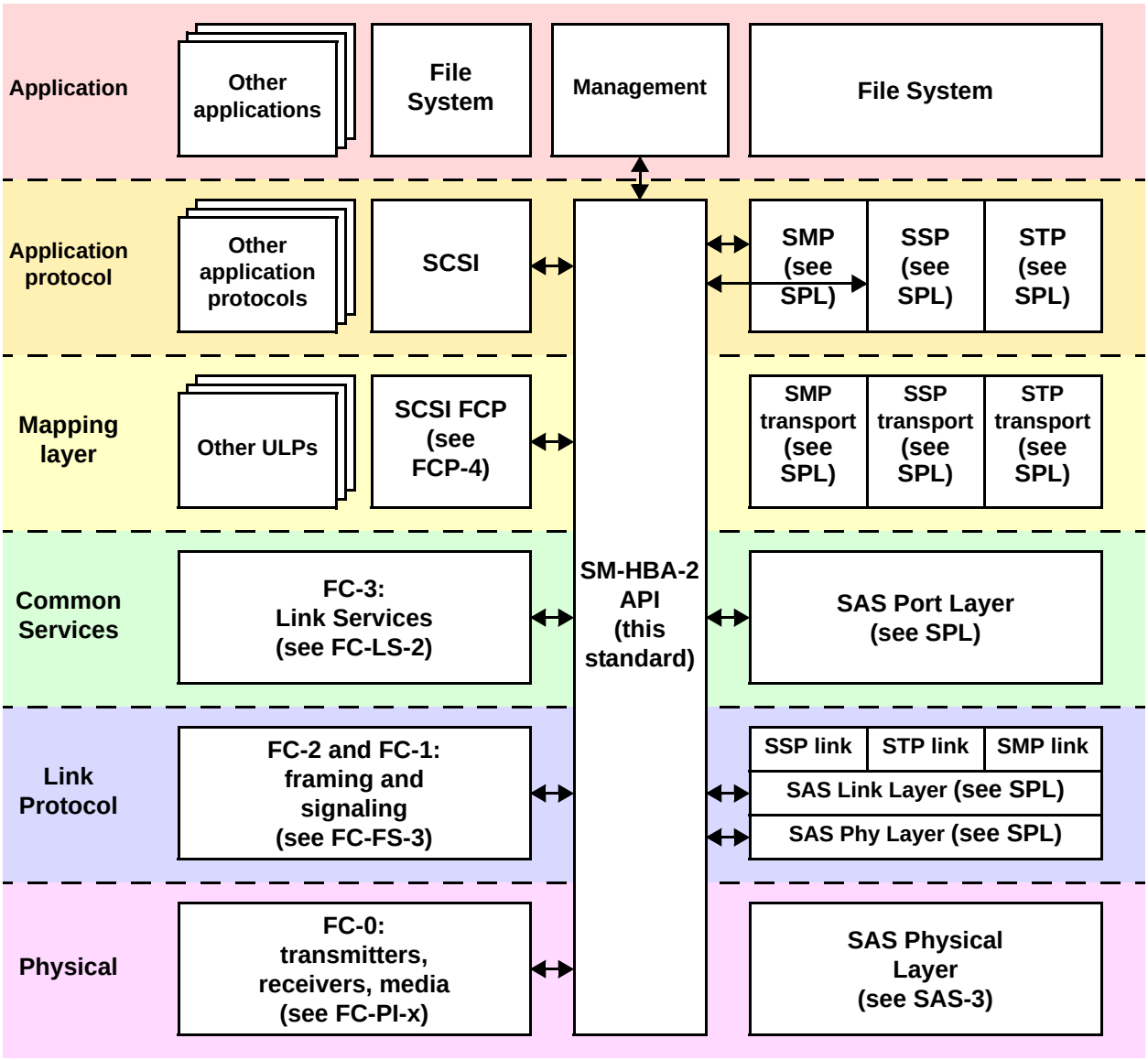

Figure 1 — Context for this standard

For standards in the Fibre Channel, Serial Attached SCSI and SCSI families of standards see the web sites of INCITS Technical Committees T10 (http://www.t10.org/) and T11 (http://www.t11.org/).

2 Normative References

2.1 Normative references

The following standards contain provisions that, by reference in the text, constitute provisions of this standard. At the time of publication, the editions listed were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

Copies of the following documents may be obtained from ANSI, an ISO member organization:

Approved ANSI standards;
approved international and regional standards (ISO and IEC); and
approved foreign standards (including JIS and DIN).

For further information, contact the ANSI Customer Service Department:

Phone: +1 212-642-4980
Web: http://webstore.ansi.org/
E-mail: storemanager@ansi.org

or the InterNational Committee for Information Technology Standards (INCITS):

Phone: 202-626-5738
Web: http://www.incits.org
E-mail: incits@itic.org

Additional availability contact information is provided below as needed.

2.2 Approved references

10GFC: ISO/IEC 14165-116:2005, *Fibre Channel - 10 Gigabit (10GFC)* [ANSI INCITS 364-2003]

FC-AL-2: ISO/IEC 14165-122, *Fibre Channel - Arbitrated Loop - 2* [ANSI INCITS 332-1999]

FC-DA: ANSI INCITS/TR-36-2004, *Fibre Channel-Device Attach*

FC-BB-5: ANSI INCITS 462-2010, *Fibre Channel - Backbone - 5*

FC-FS-2: ANSI INCITS 424-2006, *Fibre Channel - Framing and Signaling - 2*

FC-GS-6: ANSI INCITS 463-2010, *Fibre Channel-Generic Services - 6*

FC-HBA: ANSI INCITS 386-2004, *Fibre Channel - Host Bus Adapter API*

FC-LS: ANSI INCITS 433-2007, *Fibre Channel - Link Services*

FC-MI: ANSI INCITS TR-30-2002, *Fibre Channel - Methodologies for Interconnects Technical Report*

FC-PI-4: ANSI INCITS 450-2009, *Fibre Channel - Physical Interfaces - 4*

FCP-3: ANSI INCITS 416-2006, *Fibre Channel Protocol - 3*

FC-SW-5: ANSI INCITS 461-2010, *Switch Fabric - Generation 5*

FC-SWAPI: ANSI INCITS 399-2004, *Fibre Channel - Switch API*

SAM-4: ISO/IEC 14776-414, *SCSI Architecture Model - 3* [ANSI INCITS 447-2008]

SAS-2: ANSI INCITS 457-2010, *Serial Attached SCSI - 2*

SBC-2: ISO/IEC 14776-322:2007, *SCSI Block Commands - 2* [ANSI INCITS 405-2005]

SM-HBA: ANSI INCITS 428-2007, *Storage Management - Host Bus Adapter API*

SPC-3: ISO/IEC 14776-453:20095, *SCSI Primary Commands - 3* [ANSI INCITS 408-2005]

SPI-5: ISO/IEC 14776-115:2004, *SCSI Parallel Interface - 5* [ANSI INCITS 367-2003]

2.3 References under development

At the time of publication, the following referenced American National Standards were still under development. For information on the current status of the document, or regarding availability, contact the relevant standards body or other organization as shown.

FC-BB-6: INCITS Project 2159-D, *Fibre Channel - Backbone - 6*

FC-DA-2: INCITS Project 1870-DT, *Fibre Channel-Device Attach - 2*

FC-FS-3: INCITS Project 1861-D, *Fibre Channel - Framing and Signaling - 3*

FC-GS-7: INCITS Project 2204-D, *Fibre Channel - Generic Services - 7*

FC-LS-2: INCITS Project 1620-D, *Fibre Channel - Link Services - 2*

FCP-4: INCITS Project 1828-D, *Fibre Channel Protocol - 4*

FC-PI-3: INCITS Project 1625-D*, Fibre Channel - Physical Interfaces - 3*

FC-PI-5: INCITS Project 2118-D*, Fibre Channel - Physical Interfaces - 5*

SAM-5: INCITS Project 2104-D, *SCSI Architecture Model - 4*

SPL: INCITS Project 2124-D, *SAS Protocol Layer*

SAS-3: INCITS Project 2212-D, *Serial Attached SCSI - 3*

SBC-3: INCITS Project 1799-D, *SCSI Block Commands - 3*

SPC-4: INCITS Project 1731-D, *SCSI Primary Commands - 4*

2.4 IETF references

Copies of the following approved IETF standards may be obtained through the Internet Engineering Task Force (IETF) at www.ietf.org.

RFC 768: User Datagram Protocol, August 1980.

RFC 791: Internet Protocol, September 1981.

RFC 793: Transmission Control Protocol, September 1981.

RFC 2460: Internet Protocol, Version 6 (IPv6) Specification, December 1998.

2.5 Other references

For information on the current status of the listed document(s), or regarding availability, contact the indicated organization.

The SATA-2.6 document may be obtained from SATA I/O at www.sata-io.org.

SATA-2.6: Serial ATA Revision 2.6.

PCI documents may be obtained from PCI-SIG at

http://www.pcisig.com/specifications/ordering_information/.

PCI 3.0: PCI Local Bus Specification Revision 3.0.

SRIOV 1.1: PCI Single Root I/O Virtualization Specification 1.1

Copies of the following approved IEEE standards may be obtained through the Institute of Electrical and Elecontrics Engineers (IEEE) at http://standards.ieee.org

802.1Q-2005: Virtual Bridged Local Area Networks.

3 Definitions, symbols, abbreviations, and conventions

3.1 Definitions

3.1.1 address identifier An address value used to identify source (S_ID) or destination (D_ID) of a frame (see FC-FS-3).

3.1.2 application programming interface (API) A grammar within a programming language that provides means for higher-level software (i.e., applications) to control a specialized subsystem. An application programming interface may abstract a simpler uniform feature set from more complex and variant native interfaces of subsystems of similar purpose but differing implementations.

3.1.3 Arbitrated Loop: A Fibre Channel topology where L_Ports use arbitration to gain access to the loop (see FC-AL-2).

3.1.4 ASCII array: An ordered sequence of zero or more bytes, each having value equal to a Printable ASCII character (see 3.1.64). The number of bytes in an ASCII Array is determined by means external to itself.

3.1.5 ASCII string: An ordered sequence of one or more bytes, the last of which has value zero and all others have value equal to a Printable ASCII character (see 3.1.64).

3.1.6 byte: An eight-bit entity with its least significant bit denoted as bit 0 and most significant bit as bit 7. The most significant bit is shown on the left side, unless specifically indicated otherwise.

3.1.7 callback: A call to an application function previously registered for asynchronous event reporting. (see 7.9.1.1).

3.1.8 classes of service: Type of frame delivery services used by the communicating end ports that may also be supported through a fabric (see FC-FS-3).

3.1.9 Common Transport (CT): A protocol defined by (see FC-GS-7) that provides access to Services and their related Servers. CT may also refer to an instance of the Common Transport (see FC-GS-7).

3.1.10 concatenation: A logical operation that joins together strings of data and is represented with the symbol || (e.g., S_ID||X_ID represents S_ID concatenated with X_ID to provide a reference of uniqueness).

3.1.11 data frame: A Device_Data frame, a Video_Data frame, or an FC-4 Link_Data frame (see FC-FS-3).

3.1.12 Destination_Identifier (D_ID): The address identifier used to indicate the targeted destination end port of the transmitted frame (see FC-FS-3).

3.1.13 end port: An Nx_Port or a Private NL_Port or any SAS Port.

3.1.14 event: A change of condition of an object that is supported by an HBA API.

3.1.15 event category: A group of event types that affect the same kind of object and share a common registration function.

3.1.16 event type: A classification of events by the specific change of condition that occurred.

3.1.17 F_Port: The Link Control Facility within the Fabric that attaches to an N_Port through a link. An F_Port is addressable by the N_Port attached to it, with a common well-known address identifier FFFFFEh (see FC-FS-3).

3.1.18 Fabric: The entity that interconnects Nx_Ports attached to it and is capable of routing frames by using only the D_ID information in a FC-2 frame header (see FC-FS-3).

3.1.19 Fabric_Name: A Name_Identifier associated with a Fabric (see FC-FS-3).

3.1.20 FC-4 Type: An FC-4 protocol associated with the value in the Type field in the header of a data frame (see FC-FS-3).

3.1.21 FC_Port: A port that is capable of transmitting or receiving Fibre Channel frames. FC_Ports include N_Ports, NL_Ports, Nx_Ports, L_Ports, F_Ports, FL_Ports, Fx_Ports, E_Ports, B_Ports, G_Ports and GL_Ports (see FC-FS-3).

3.1.22 FCP_Port An end port that supports the SCSI Fibre Channel Protocol (see FCP-4).

3.1.23 FCP-4 A SCSI transport protocol for using Fibre Channel as a SCSI service delivery subsystem (see FCP-4).

3.1.24 FCP-4 target port A SCSI target port for which the transport protocol is FCP-4 (see FCP-4).

3.1.25 FL_Port: An F_Port that contains Arbitrated Loop functions associated with Arbitrated Loop topology (see FC-AL-2 and FC-SW-5).

3.1.26 frame: An indivisible unit of information used by FC-2 (see FC-FS-3).

3.1.27 Fx_Port: A switch port capable of operating as an F_Port or FL_Port (see FC-FS-3).

3.1.28 Generic Services: The collection of Services defined by (see FC-GS-7).

3.1.29 HBA API instance Code implementing an HBA API library that is a component of an application that is operating on a computer system.

3.1.30 HBA API library A library of function calls that presents an HBA API compliant with this standard.

3.1.31 HBA specific library HBA specific software structured as a library of function calls compliant with the requirements of this standard.

3.1.32 HBA specific software A component of an HBA API library that adapts some vendor specific category of HBAs and their drivers to software that presents an HBA API compliant with this standard.

3.1.33 host bus adapter (HBA) A hardware component together with its supporting software that provides an interface from an operating system to one or more links.

3.1.34 Internet Protocol (IP): A protocol for communicating data packets between identified endpoints on a multipoint network. (IPv4 see RFC 791 and IPv6 see RFC 2460).

3.1.35 IP Address: An identifier of an endpoint in Internet Protocol.

3.1.36 link: A physical communication medium between two end points and its associated transmitters and receivers

3.1.37 local end port: An end port on the same system with respect to an HBA API instance.

3.1.38 logical unit An externally addressable entity within a target that implements a SCSI device model and contains a device server (see SAM-5).

3.1.39 logical unit number (LUN) An encoded 64-bit identifier for a logical unit (see SAM-5).

3.1.40 Logical Unit Unique Identifier (LUID): An Identification Descriptor from the Vital Products Data Device Identification Page (Page 83h) returned by a logical unit in reply to a SCSI INQUIRY command (see SPC-4) with further constraints specified in 6.10.2.5

3.1.41 Loop Initialization Primitive (LIP): Any one of the Primitive Sequences used to cause initialization of all L_Ports attached to an Arbitrated Loop topology (see FC-AL-2).

3.1.42 L_Port: An FC_Port that contains Arbitrated Loop functions associated with Arbitrated Loop topology (see FC-AL-2).

3.1.43 megabyte: 1 000 000 bytes.

3.1.44 Name Server: A Server among those provided by Generic Services (see FC-GS-7).

3.1.45 Name_Identifier A 64-bit identifier, with a 60-bit value preceded by a 4-bit Network_Address_Authority Identifier (NAA), used to identify entities in Fibre Channel (e.g., N_Port, node, F_Port, or Fabric) (see FC-FS-3).

3.1.46 NL_Port: An N_Port that contains the Loop Port State Machine defined in FC-AL-2 (see FC-AL-2). It may be attached to one or more NL_Ports and FL_Ports in an Arbitrated Loop topology. Without the qualifier Public or Private, an NL_Port shall be a Public NL_Port.

3.1.47 node: A collection of one or more end ports controlled by a level above FC-2 (see FC-FS-3).

3.1.48 Node_Name A Name_Identifier associated with a node (see FC-FS-3).

3.1.49 Node Symbolic Name: A Symbolic Name associated with a node (see FC-GS-7).

3.1.50 Not_Operational Primitive Sequence (NOS) A primitive sequence indicating that an FC_Port is in the nonoperational state (see FC-FS-3).

3.1.51 N_Port: A hardware entity that includes a Link Control Facility but not Arbitrated Loop functions associated with Arbitrated Loop topology, and has the ability to act as an Originator, a Responder, or both. Well-known addresses are considered to be N_Ports (see FC-FS-3 and FC-AL-2).

3.1.52 N_Port_ID A Fabric unique address identifier by which an N_Port is known. The identifier may be assigned by the fabric during the initialization procedure or by other procedures not defined in this standard. The identifier is used in the S_ID and D_ID fields of a frame (see FC-FS-3).

3.1.53 N_Port_Name A Name_Identifier associated with an Nx_Port or a Private NL_Port (see FC-FS-3).

3.1.54 Nx_Port: An FC_Port capable of operating as an N_Port or Public NL_Port, but not as a Private NL_Port. By use of the term Nx_Port, this standard neither specifies nor constrains the behavior of Private NL_Ports (see FC-FS-3 and FC-AL-2).

3.1.55 operating system (OS): Software running on a system that interposes between the physical resources of the system and the application programs using it, abstracting the behavior of the resources and arbitrating access to them

3.1.56 Ordered Set A transmission word composed of a special character in its first (i.e., left-most) position and data characters in its remaining positions (see FC-FS-3).

3.1.57 Payload: Contents of the Data Field of a frame, excluding Optional Headers and fill bytes, if present (see FC-FS-3).

3.1.58 persistent binding A function of an HBA that retains a pairing of an OS SCSI identification and a SCSI transport protocol identification across resets of the HBA, its fabric, or its OS, and subsequently reestablishes a target mapping based on the pairing, or else a representation of a single such pairing.

3.1.59 Platform: An association of one or more Nodes for the purpose of discovery and management (see FC-GS-7).

3.1.60 Port_Identifier: A SAS Port Identifier (see SPL).

3.1.61 Port Symbolic Name: A Symbolic Name associated with an FC_Port (see FC-GS-7).

3.1.62 Primitive Sequence: An Ordered Set transmitted repeatedly and continuously until a specified response is received (see FC-FS-3).

3.1.63 Primitive Signal: An Ordered Set designated to have a special meaning. (e.g., an Idle or R_RDY) (see FC-FS-3).

3.1.64 Printable ASCII Character: A character in the range 20h through 7Eh.

3.1.65 Private NL_Port An NL_Port that does not attempt a Fabric Login and does not transmit the primitive signal OPN(00,x) (see FC-AL-2).

3.1.66 Public NL_Port An NL_Port that attempts a Fabric Login (see FC-AL-2).

3.1.67 SAS Serial Attached SCSI (see SPL).

3.1.68 SAS dword A sequence of four contiguous bytes or four contiguous characters considered as a unit.

3.1.69 SAS Port A SAS Port (see SPL).

3.1.70 SCSI target device A SCSI device containing logical units and SCSI target ports that receives device service and task management requests for processing (see SAM-5).

3.1.71 SCSI target port A SCSI target device object that acts as the connection between device servers and task managers and the service delivery subsystem through which requests and responses are routed (see SAM-5).

3.1.72 Server: A Server is an entity that accepts CT requests and provides CT responses. A Server is accessed via a Service (e.g., the Name Server is accessed using the Directory Service) (see FC-GS-7).

3.1.73 service: A service is provided by a Node, accessible via an N_Port that is addressed by a Well-known Address or an N_Port_ID (e.g., the Directory Service and the Alias Service). A service provides access to one or more Servers (see FC-GS-7).

3.1.74 SMP Serial Management Protocol (see SPL).

3.1.75 Source_Identifier (S_ID): The address identifier used to indicate the source end port of the transmitted frame (see FC-FS-3).

3.1.76 SSP Serial SCSI Protocol (see SPL).

3.1.77 SSP_Port A SSP Port (see SPL).

3.1.78 STP Serial ATA Tunneling Protocol (see SPL).

3.1.79 storage area network (SAN) A data communication system the primary or only purpose of which is providing access from computer systems to SCSI target devices. In the context of this standard, a storage area network is always implemented with Fibre Channel technology.

3.1.80 Symbolic Name: A user-defined name for an object, composed of Printable ASCII Characters (see 3.1.64). Uniqueness of its value is not required.

3.1.81 target Synonymous with SCSI target port

3.1.82 target mapping A function of an HBA that makes an OS SCSI identification of a target or logical unit operationally equivalent to a specified SCSI transport protocol identification of a target or logical unit; or else a representation of a single such equivalence.

3.1.83 TCP Port Number: An identifier of a destination in Transmission Control Protocol (see RFC 793).

3.1.84 Transmission Control Protocol (TCP): A protocol communicating reliable flow-controlled byte streams over Internet Protocol allowing independent concurrent streams to multiple destinations at any IP Address (see RFC 793).

3.1.85 transmission word: A string of four contiguous transmission characters occurring on boundaries that are zero modulo 4 from a previously received or transmitted special character (see FC-FS-3).

3.1.86 UDP Port Number: An identifier of a destination in User Datagram Protocol (see RFC 768).

3.1.87 User Datagram Protocol (UDP): A protocol communicating a packet stream with no incremental reliability over Internet Protocol allowing multiple independent concurrent destinations at any IP Address (see RFC 768).

3.1.88 vendor specific library Obsolete term for HBA specific library (see FC-MI).

3.1.89 Well-known addresses: A set of address identifiers defined in this standard to access global Server functions. (e.g., a Name Server) (see FC-FS-3).

3.1.90 word: A string of four contiguous bytes occurring on boundaries that are zero modulo 4 from a specified reference.

3.1.91 Worldwide_Name (WWN): A Name_Identifier that is worldwide unique, and represented by a 64-bit value (see FC-FS-3).

3.1.92 Worldwide Node Name (WWNN): A Name_Identifier referencing a node that is worldwide unique, and represented by a 64-bit value. Same as Node_Name (see FC-FS-3).

3.1.93 wrapper library A component of an HBA API library that combines the interfaces of one or more HBA specific libraries into a single interface compliant with this standard.

3.2 Symbols and abbreviations

Symbol / Abbreviation	Meaning
±	plus or minus
x	multiply
+	add
-	subtract
||	concatenate
= or EQ	equal
≈	approximately equal
≠ or NE	not equal
< or LT	less than
≤ or LE	less than or equal to
> or GT	greater than
≥ or GE	greater than or equal to
API	application programming interface
CT	Common Transport (see FC-GS-7)
ELS	Extended Link Service (see FC-LS-2)
FC	Fibre Channel
FC-PI-x	the current set of FC physical layer standards (see FC-PI-3, FC-PI-5, and 10GFC)
HBA	host bus adapter
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LSB	least significant bit
LUID	Logical Unit Unique Identifier
LUN	logical unit number
LIP	Loop Initialization Primitive Sequence
μsec	microsecond (i.e., 10^{-6} second)
NOS	Not_Operational Primitive Sequence
OS	operating system
SCSI	Small Computer System Interface (see SAM-5)
SAM-4	SCSI Architecture Model-4 (see SAM-5)
SAN	storage area network
SPC-4	SCSI Primary Commands-4 (see SPC-4)

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WWN	Worldwide_Name
WWNN	Worldwide Node Name

3.3 Keywords

3.3.1 expected A keyword used to describe the behavior of the hardware or software in the design models presumed by this standard. Other hardware and software design models may also be implemented.

3.3.2 invalid A keyword used to describe an illegal or unsupported bit, byte, word, field or code value. Receipt of an invalid bit, byte, word, field or code value shall be reported as an error.

3.3.3 mandatory A keyword indicating an item that is required to be implemented as defined in this standard to claim compliance with this standard.

3.3.4 may A keyword that indicates flexibility of choice with no implied preference.

3.3.5 may not Keywords that indicates flexibility of choice with no implied preference.

3.3.6 obsolete A keyword indicating that an item was defined in prior standards but has been removed from this standard.

3.3.7 optional A keyword that describes features that are not required to be implemented by this standard. However, if any optional feature defined by this standard is implemented, it shall be implemented as defined in this standard.

3.3.8 reserved A keyword referring to bits, bytes, words, fields and code values that are set aside for future standardization. Their use and interpretation may be specified by future extensions to this or other standards. A reserved bit, byte, word or field shall be set to zero, or in accordance with a future extension to this standard. Recipients are not required to check reserved bits, bytes, words or fields for zero values. Receipt of reserved code values in defined fields shall be reported as an error.

3.3.9 shall A keyword indicating a mandatory requirement. Designers are required to implement all such requirements to ensure interoperability with other products that conform to this standard.

3.3.10 should A keyword indicating flexibility of choice with a preferred alternative; equivalent to the phrase "it is recommended".

3.4 Conventions

Certain words and terms used in this American National Standard have a specific meaning beyond the normal English meaning. These words and terms are defined either in clause 3 or in the text where they first appear.

Fields containing only one bit are usually referred to as the name bit instead of the name field.

Numbers that are not immediately followed by lower-case b or h are decimal values.

Numbers immediately followed by lower-case b (xxb) are binary values.

Numbers immediately followed by lower-case h (xxh) are hexadecimal values.

Hexadecimal digits that are alphabetic characters are upper case (i.e., ABCDEF, not abcdef).

Hexadecimal numbers may be separated into groups of four digits by spaces. If the number is not a multiple of four digits, the first group may have fewer than four digits (e.g., AB CDEF 1234 5678h)

Decimal fractions are initiated with a dot (e.g., two and one half is represented as 2.5).

Decimal numbers having a value exceeding 999 are separated with a space(s) (e.g., 24 255).

An alphanumeric list (e.g., a, b, c or A, B, C) of items indicate the items in the list are unordered.

A numeric list (e.g., 1, 2, 3) of items indicate the items in the list are ordered (i.e., item 1 shall occur or complete before item 2).

In the event of conflicting information the precedence for requirements defined in this standard is

1) text,
2) tables, then
3) figures.

3.5 Notation for Procedures and Functions

Procedures and functions are specified in the syntax of the "C" programming language.

4 General Constraints

4.1 Software Structure

The API specified in this standard shall be presented in the form of a library of function calls specified in this standard. In OS environments that have a well-defined software structure for a library of function calls, that structure shall be used.

An HBA API that is compliant with this standard facilitates common management methodologies for configurations of FC and SAS HBAs that may be provided by multiple vendors and installed or removed at various times. This shall be achieved by a software structure that is either OS specific and documented by the OS vendor (see 5.2.1) or as defined in this standard for an OS independent structure (see 5.2.2).

Should a documented OS structure not be used, a software structure is specified in this standard that is composed of a single API for access to all HBAs (i.e., a wrapper library) layered on top of a configurable set of modules each of which provides service for some collection of vendor specific HBAs and their drivers (i.e., HBA specific libraries).

The relationships of the components of either software structure to one another and to their environment is shown in figure 2. Also shown in figure 2 is that both the API offered by the wrapper library to applications and the API offered by HBA specific libraries to the wrapper library are specified by this standard. They are identical interfaces other than as indicated in this standard.

A wrapper library or OS-specific library that is compliant with this standard may be identified as an SM-HBA-2 compliant HBA API. A wrapper library that is compliant with this standard may be identified as an SM-HBA-2 compliant wrapper library and may also be identified as an SM-HBA-2 compliant HBA API. An HBA specific library that is compliant with this standard may be identified as an SM-HBA-2 compliant HBA specific library but it shall not be identified as an SM-HBA-2 compliant HBA API.

4.2 Backwards Compatibility

A library compliant with this standard may also provide interfaces compliant with one or more predecessor standard HBA APIs (e.g., SM-HBA, FC-HBA, and FC-MI). This standard specifies methods for registration of compliant libraries that distinguish them from libraries compliant with predecessor standard HBA APIs (see clause 8).

4.3 C language

This standard defines an API only in the C language. Functionally equivalent APIs may be implemented in other languages but these shall not be referenced as SM-HBA-2 compliant. In this standard, references within C code declarations to the names of other C code declarations shall be considered normative specifications that the structure and semantics of the element referencing the name shall be as specified in the declaration that is named.

All functions provided in compliance with function specifications in this standard shall use C-style calling conventions. This constraint does not limit the internal implementation of components of an HBA API library.

Unless specified otherwise in this standard, data structures and elements shall be stored in memory as determined by the local machine, operating system, and C compiler.

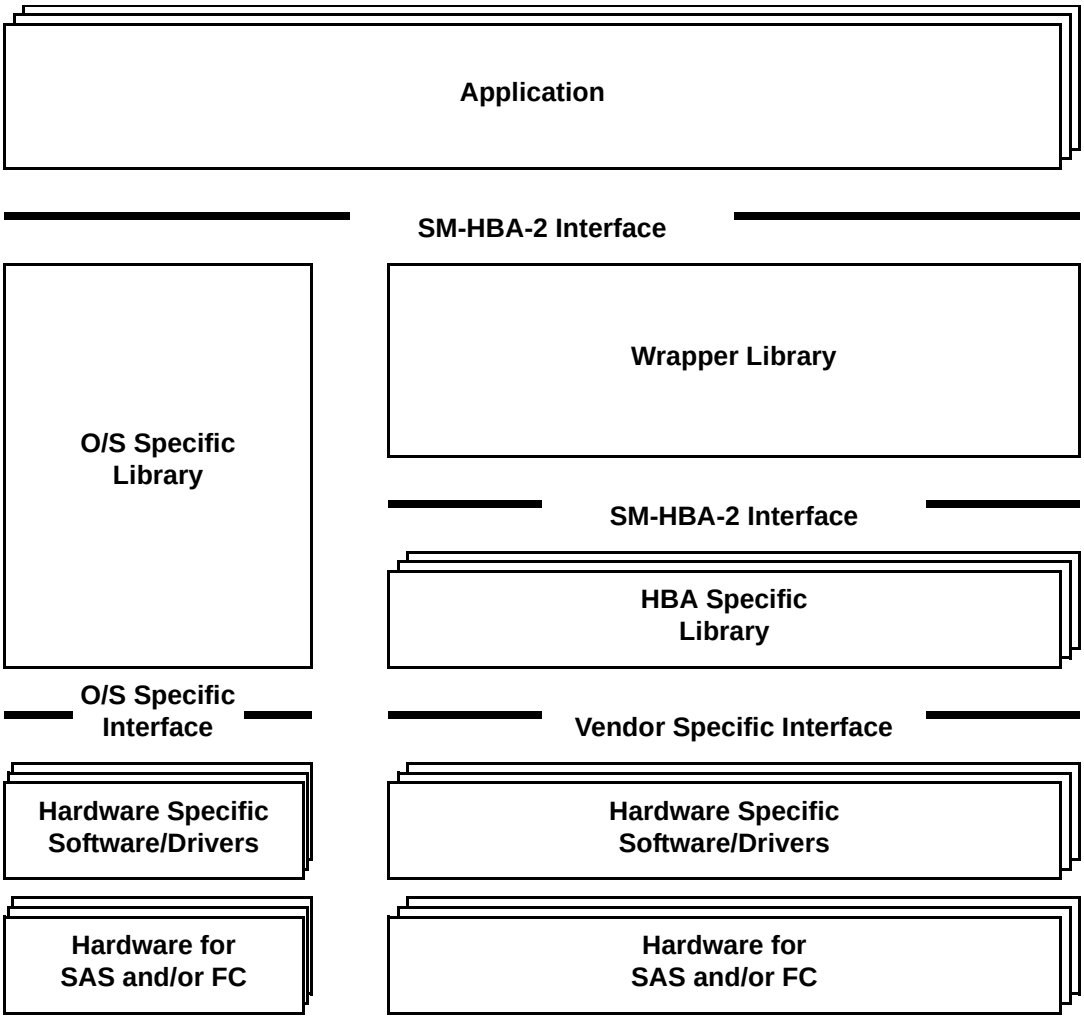

Figure 2 — SM-HBA-2 system architecture

This standard provides declarations for all data structures that it requires. Although these declarations may in common practice be combined into a C header file, that is not required for compliance.

4.4 Operating System Dependencies

Although this standard has been written with attention to making it independent of specific operating environments, it has in some cases been necessary to make normative statements specific to certain OSs in order to assure interoperability. Normative statements specific to an OS shall not be considered in determining the compliance of implementations for other OSs. This may lead to less assurance of interoperability of compliant components in OSs for which specific normative statements have not been made.

5 Software Structure and Behavior

5.1 Overview

This clause specifies constraints on the overall structure and behavior of an implementation compliant with this standard.

This standard facilitates the implementation of a uniform and unitary interface to HBAs produced by multiple vendors that may be installed in the same system. An HBA API may be composed of component modules developed by independent vendors. This standard specifies not only the external behavior of an HBA API but also certain internal structure and interfaces that represent boundaries of likely modularization.

This standard recognizes that an implementation may run in the context of an operating system that enables both concurrent and serial operation of related software applications. This clause specifies certain expectations for consistency of results in multitasking environments.

5.2 Software Structure

5.2.1 OS specific structure

An OS specific structure for an HBA API library shall meet these criteria:

a) The HBA API shall be presented in the form of a library of function calls that includes the function calls specified as required for an HBA API library in this standard;
b) All functions in an HBA API library that have the same name as a function in this standard shall comply with all requirements of this standard for the function;
c) The software that interfaces to HBA specific software and presents the HBA API shall be available from the OS vendor;
d) Documentation shall be available from the OS vendor that fully specifies the structure and interface for HBA specific software;
e) The OS specific structure shall allow adding and removing individual HBAs and HBA specific software without affecting other HBAs and other HBA specific software, other than any temporarily degraded operation of the OS necessary for adding or removing an HBA;
f) If the OS allows physical or logical addition or removal of HBAs without requiring the OS to pass through a state of degraded operation, the OS specific structure for the HBA API shall allow addition or removal of the HBA specific software for the HBAs without requiring the OS to pass through a state of degraded operation;
g) Documentation shall be available from the OS vendor that fully specifies the means for adding and removing individual HBAs and HBA specific software without affecting the operation of other HBAs and other HBA specific software; and
h) Any necessary initialization beyond that inherent in loading the library shall be implemented in the function HBA_LoadLibrary.

5.2.2 OS independent structure

If an OS specific structure is not used, this standard defines a single C-style library interface that shall be implemented at two levels. At the upper level, a wrapper library shall provide the HBA API specified in this standard to applications and shall provide the ability to handle multiple vendor implementations of the HBA API through dynamic loading of HBA specific libraries. The functions of the wrapper library shall invoke their respective functions in HBA specific libraries provided by each HBA vendor. The relationships of the modules implementing these levels with one another and with other software are shown in figure 2. There

is a one to one correspondence between the functions of the wrapper library and the functions of the HBA specific libraries except as noted in clause 7.

Implementations shall not preclude multiple instances of the wrapper library in an OS (e.g., for 32 versus 64 bit operation, or for vendor-specialized versions). The unitary interface goal may be compromised if there is more than a single instance of the wrapper library.

Initialization of libraries shall be implemented in the function HBA_LoadLibrary (see 7.2.2). The wrapper library function HBA_LoadLibrary shall accomplish configuration determination, OS specific library linking functions, and API initialization. The HBA specific library function HBA_LoadLibrary shall only accomplish initialization of the HBA specific library in which it resides.

References to the functions of the HBA specific libraries are loaded into data structures owned by the wrapper library. HBA specific libraries accomplish this by supporting the function SMHBA2_RegisterLibrary (see 7.2.4).

5.3 Persistence of Identity

Handles are used in this API as a generic way to reference an HBA, Port, Controller, Fabric and/or Phy. The use of a handle is specified to be independent of the operating system. Each HBA, Port, Controller, Fabric and/or Phy that may be managed shall have a handle. The handle has the following characteristics:

a) the handle shall be persistent once returned;
b) if the HBA API library has OS independent structure, the low order 16 bits of the handle shall be determined by the HBA specific library uniquely among all HBAs, Ports, Controllers, Fabrics and/or Phys supported by the same HBA specific library;
c) if the HBA API library has OS independent structure, the high order 16 bits of the handle shall be determined by the wrapper library uniquely for all HBA specific libraries registered with the wrapper library;
d) the handle may change when the HBA is replaced; and
e) the handle may change across reboots;

These operations may be recognized in any manner:

a) addition of a new HBA;
b) removal of an HBA; and
c) replacement of an HBA.

When an HBA is replaced, there shall be no change in functionality, Name_Identifier, or any other HBA properties. Any change in the properties, including Name_Identifier, should be treated as addition of a new HBA.

5.4 HBA Configuration Rediscovery Effect on the API

5.4.1 Introduction

The HBA API specified by this standard has several functions in which logical or physical SAN resources are identified by an index. Use of an index to select a resource implies that the implementation maintains a table of the resources.

NOTE 1 - These tables are only logically implied and are not constrained by this standard.

These tables include the collection of HBAs available via the HBA API, the collection of end ports on an HBA, and the collection of discovered FC_Ports for an end port. In a dynamic SAN, resources may be

added or removed. This may cause reassignment of indexes in implicit tables. An application may extract and maintain its own information about SAN resources, keyed to the API's implicit tables by the index. Changing the index assignment in an implicit table without coordination may cause the information in the application's space to refer to different resources than the information in the API. This standard defines several approaches to maintaining index consistency between applications and the API, including static tables, asynchronous event reporting, and reference by Name_Identifier.

Static implicit tables do not change. Although static tables maintain consistency between the application and the HBA library, both may still become inconsistent with the actual SAN. For an application to remain current, frequent polling of the entire Name Server is necessary. To reduce the need for polling, there are two extensions that are consistent with the static table assumption:

a) the HBA_STATUS_ERROR_STALE_DATA error indicates that static tables are out of date (see 5.4.2); and
b) semistatic tables may grow without changing existing index assignments (see 5.4.3).

Asynchronous event reporting provides a reliable and timely notification of pending configuration changes.

Functions that identify objects by Name_Identifiers do not rely on indexes into implicit tables and shall return current information on the named resource independent of changes in its index.

5.4.2 HBA_STATUS_ERROR_STALE_DATA

To prevent the application from unknowingly using stale data without changing the static table design, a library shall return HBA_STATUS_ERROR_STALE_DATA for any function that references an index into an implicit table with a pending change of indexing for the calling application. This error is indicated to the application at the first attempt by the application to access the stale data.

HBA API libraries that implement the semistatic table model (see 5.4.3) do not cause changes of table indexes and shall not return HBA_STATUS_ERROR_STALE_DATA.

Functions that identify objects by Name_Identifiers rather than indexes into implicit tables shall not return HBA_STATUS_ERROR_STALE_DATA.

5.4.3 Semistatic table model

The semistatic table model preserves the relationship between SAN resources and the indexes by which the API references them but allows addition and removal of resources.

A resource that is no longer available shall continue to be assigned its index, but any function that references the index shall return HBA_STATUS_ERROR_UNAVAILABLE. A newly discovered resource shall be assigned the smallest unassigned index. Calls that identify the number of resources of a specific type shall return the largest index assigned to that type of resource, even if some indexes are assigned to resources that no longer exist. The number of resources and the size of the table may change. The following rules shall constrain such changes:

a) HBA handles shall continue to reference the same HBA even if the index is no longer installed.
b) An HBA index assigned to an HBA for which the bus position, Name_Identifier, and OS device name have not changed shall remain assigned to the same HBA even if the HBA is removed and reinstalled.
c) Handles, and HBA indexes assigned to HBAs that have been removed and not replaced shall not be reassigned. References to them shall generate HBA_STATUS_ERROR_UNAVAILABLE.

In systems that contain HBAs from multiple vendors and allow dynamic HBA reconfiguration, it it is not required that the wrapper library assign contiguous HBA indexes to HBAs from the same vendor.

5.5 Multiuse considerations

Multiple unrelated applications of the HBA API specified by this standard may operate concurrently and without coordination. In environments supporting multithreaded applications, multiple threads of the same application may similarly operate concurrently and without coordination. Although the result of any HBA API call may be affected by concurrently operating HBA API calls or events elsewhere in the fabric, the HBA API libraries shall be implemented so as not to become internally inconsistent as a result of concurrent use or external events. Implementations of the HBA API library and of wrapper libraries and HBA specific libraries if used shall prevent re-entrant operation that compromises consistent functionality.

Applications of the HBA API specified by this standard may be presented with certain concurrency issues. They may operate concurrently with other such applications, so the results of any given call may not be predictable based on any information gained from prior calls by the same application. A successfully registered event callback function may be invoked before its registration function returns to the application.

6 Attributes and Data Structures

6.1 Basic Attribute Types

```
typedef unsigned char HBA_UINT8;   /* An 8 bit unsigned integer */

typedef unsigned short HBA_UINT16; /* A 16 bit unsigned integer */

typedef unsigned int HBA_UINT32;   /* A 32 bit unsigned integer */

typedef int HBA_INT32;             /* A 32 bit signed integer */

typedef long HBA_INT64;            /* A 64 bit signed integer; */
                                   /* may use OS-specific typedef */

typedef unsigned long HBA_UINT64;  /* A 64 bit unsigned integer; */
                                   /* may use OS-specific typedef */

typedef HBA_UINT8 HBA_BOOLEAN;     /* A single true/false flag */

typedef HBA_UINT32 HBA_HANDLE;     /* handle used to identify an HBA, Port, Controller, */
                                   /* Fabric and/or Phy */

typedef struct SMHBA_HandleList {
       HBA_UINT32           NumberOfEntries;
       HBA_HANDLE           Handle[1];
} SMHBA_HANDLELIST, *PSMHBA_HANDLELIST;   /* A list of HBA handles */

typedef struct HBA_wwn {HBA_UINT8 wwn[8];} HBA_WWN, *PHBA_WWN;
                                   /*An FC-FS-2 Name_Identifier*/
                                   /* The first byte of the Name_Identifier */
                                   /* as transmitted on a link */
                                   /* (i.e., the most significant byte of word 0) */
                                   /* shall be in the first byte of the array, */
                                   /* and successive bytes of the Name_Identifier */
                                   /* as transmitted on a link */
                                   /* shall be in successive bytes of the array. */

typedef struct SMHBA_scsilun {HBA_UINT8 lun[8];} SMHBA_SCSILUN, *PSMHBA_SCSILUN;
                                   /* A byte array representation of a SCSI */
                                   /* LUN (see SAM-5). The first byte of the */
                                   /* LUN shall be in the first byte of the */
                                   /* array, and successive bytes of the SCSI */
                                   /* LUN shall be in successive bytes of the */
                                   /* array. */

typedef struct SMHBA2_MACAddr {HBA_UINT8 macbyte[8];} SMHBA2_MACADDR, *PSMHBA2_MACADDR;
                                   /* An IEEE 802 MAC Address. The first byte of */
                                   /* the MAC Address as transmitted on a link */
                                   /* shall be in the first byte of the array,*/
                                   /* and successive bytes of the MAC Address as*/
                                   /* transmitted on a link shall be in successive */
                                   /* bytes of the array. The final two bytes of the */
                                   /* array are reserved. */
```

6.2 Status Return Values

Functions that return an object of type HBA_STATUS shall set the value of that object to a value defined in this subclause.

```
typedef       HBA_UINT32    HBA_STATUS;          /* Function status return structure */

/* No Error */
#define       HBA_STATUS_OK                             0

/* Error */
#define       HBA_STATUS_ERROR                          1

/* Function not supported.*/
#define       HBA_STATUS_ERROR_NOT_SUPPORTED            2

/* invalid handle */
#define       HBA_STATUS_ERROR_INVALID_HANDLE           3

/* Bad argument */
#define       HBA_STATUS_ERROR_ARG                      4

/* Name_Identifier not recognized */
#define       HBA_STATUS_ERROR_ILLEGAL_WWN              5

/* Index not recognized */
#define       HBA_STATUS_ERROR_ILLEGAL_INDEX            6

/* Larger buffer required */
#define       HBA_STATUS_ERROR_MORE_DATA                7

/* obsolete                                             8 */

/* SCSI Check Condition reported */
#define       HBA_STATUS_SCSI_CHECK_CONDITION           9

/* HBA busy or reserved, retry may be effective */
#define       HBA_STATUS_ERROR_BUSY                     10

/* Request timed out, retry may be effective */
#define       HBA_STATUS_ERROR_TRY_AGAIN                11

/* Referenced HBA has been removed or deactivated */
#define       HBA_STATUS_ERROR_UNAVAILABLE              12

/* The requested ELS was rejected by the local HBA */
#define       HBA_STATUS_ERROR_ELS_REJECT               13

/* The specified LUN is not provided by the specified HBA */
#define       HBA_STATUS_ERROR_INVALID_LUN              14

/* An incompatibility has been detected */
/* among the library and driver modules invoked */
/* that may cause one or more functions */
/* in the highest version they all support */
/* to operate incorrectly. */
/* The differing function sets of software modules */
/* implementing different versions of the HBA API specification */
```

```
/* does not in itself constitute an incompatibility.*/
/* Known interoperability bugs among supposedly compatible versions */
/* should be reported as incompatibilities, */
/* but not all such interoperability bugs may be known. */
/* This value may be returned by any function */
/* that calls an HBA specific library and returns an HBA_STATUS, */
/* and by HBA_LoadLibrary. */
#define       HBA_STATUS_ERROR_INCOMPATIBLE             15

/* Multiple adapters have a matching Name_Identifier. */
/* This may occur if the NodeWWN of multiple adapters is identical */
/* or if the Port_Identifier of multiple SAS ports is identical. */
#define       HBA_STATUS_ERROR_AMBIGUOUS_WWN            16

/* A persistent binding request included a bad local SCSI bus number */
#define       HBA_STATUS_ERROR_LOCAL_BUS                17

/* A persistent binding request included a bad local SCSI target number */
#define       HBA_STATUS_ERROR_LOCAL_TARGET             18

/* A persistent binding request included a bad local SCSI logical unit number */
#define       HBA_STATUS_ERROR_LOCAL_LUN                19

/* A persistent binding set request included */
/* a local SCSI ID that was already bound */
#define       HBA_STATUS_ERROR_LOCAL_SCSIID_BOUND       20

/* A persistent binding request included a bad or unlocatable FCP-4 target FCID */
#define       HBA_STATUS_ERROR_TARGET_FCID              21

/* A persistent binding request included a bad FCP-4 target Node_Name */
#define       HBA_STATUS_ERROR_TARGET_NODE_WWN          22

/* A persistent binding request included a bad FCP-4 target N_Port_Name */
#define       HBA_STATUS_ERROR_TARGET_PORT_WWN          23

/* A persistent binding request included */
/* an FCP logical unit number not defined by the identified target*/
#define HBA_STATUS_ERROR_TARGET_LUN                     24

/* A persistent binding request included */
/* an undefined or otherwise inaccessible Logical Unit Unique Identifier */
#define       HBA_STATUS_ERROR_TARGET_LUID              25

/* A persistent binding remove request included */
/* a binding that did not match a binding established by the specified local end port */
#define       HBA_STATUS_ERROR_NO_SUCH_BINDING          26

/* A SCSI command was requested to an end port that was not a SCSI target Port */
#define       HBA_STATUS_ERROR_NOT_A_TARGET             27

/* A request was made concerning an unsupported FC-4 protocol */
#define       HBA_STATUS_ERROR_UNSUPPORTED_FC4          28

/* A request was made to enable unimplemented capabilities for a local end port */
#define       HBA_STATUS_ERROR_INCAPABLE                29

/* A SCSI function was rejected to prevent causing */
/* a SCSI overlapped command condition (see SAM-5) */
```

```
#define       HBA_STATUS_ERROR_TARGET_BUSY              30

/* A call was made to HBA_FreeLibrary when no library was loaded */
#define       HBA_STATUS_ERROR_NOT_LOADED               31

/* A call was made to HBA_LoadLibrary when a library was already loaded */
#define       HBA_STATUS_ERROR_ALREADY_LOADED           32

/* The Address Identifier specified in a call to HBA_SendRNIDV2 */
/* violates access control rules for that call */
#define       HBA_STATUS_ERROR_ILLEGAL_FCID             33

/* A device was identified in a SCSI-specific call that is not a SCSI device */
#define       HBA_STATUS_ERROR_NOT_ASCSIDEVICE          34

/* The code used to identify an FC-4 protocol is not a code specified by this standard */
#define       HBA_STATUS_ERROR_INVALID_PROTOCOL_TYPE    35

/* The code used to identify an event type was not a code specified by this standard */
#define       HBA_STATUS_ERROR_BAD_EVENT_TYPE           36

/* A PHY was identified in a PHY type-specific call that is not the right PHY type */
#define       HBA_STATUS_ERROR_INCORRECT_PHY_TYPE       37
```

6.3 Adapter Attributes

6.3.1 Generic Adapter

6.3.1.1 Generic Adapter Requirements

Requirements are specified in annex A for support of the attributes specified in 6.3.1.

6.3.1.2 Generic Adapter Attribute Data Declarations

Any data object of type SMHBA2_HBAOPTIONS shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_HBAOPTIONS;
```

Any data object of type SMHBA2_HBAOPTIONS shall have a value that is the bit-wise OR of one or more values defined in this list:

```
#define       SMHBA2_HBA_VA_SPT    (1<<0) /* Adapter Virtualization supported */
#define       SMHBA2_HBA_ENA       (1<<1) /* Adapter Virtualization enabled*/
#define       SMHBA2_VHBA          (1<<2) /* This is a Virtual Adapter */
```

Any data object of the type SMHBA2_ADAPTERATTRIBUTES shall have the format defined in this structure:

```
typedef struct SMHBA2_AdapterAttributes {
       HBA_HANDLE           HBAHandle;
       SMHBA2_HBAOPTIONS    HBAOptions;
       char                 Manufacturer[64];
       char                 SerialNumber[64];
       char                 Model[256];
       char                 ModelDescription[256];
       char                 HardwareVersion[256];
       char                 DriverVersion[256];
       char                 OptionROMVersion[256];
```

```
       char                FirmwareVersion[256];
       HBA_UINT32          VendorSpecificID;
       char                DriverName[256];
       char                HBASymbolicName[256];
       char                RedundantOptionROMVersion[256];
       char                RedundantFirmwareVersion[256];
} SMHBA2_ADAPTERATTRIBUTES, *PSMHBA2_ADAPTERATTRIBUTES;
```

6.3.1.3 Generic Adapter Attribute Specifications

6.3.1.3.1 HBAHandle

HBAHandle is an integer that uniquely identifies the HBA among all HBAs within an instance of the HBA API software. The value of HBAHandle shall not change once it is assigned (e.g., at power-up, or hot-plug of an Adapter), and shall not be reassigned to a different HBA unless by reinitialization of the HBA API software.

6.3.1.3.2 HBAOptions

If adapter virtualization is supported on this physical HBA, then SMHBA2_HBA_VA_SPT bit-wise ORed with the value of HBAOptions shall be one. If adapter virtualization is not supported on this physical HBA, or if this is not a physical HBA, then SMHBA2_HBA_VA_SPT bit-wise ORed with the value of HBAOptions shall be zero.

If adapter virtualization is enabled on this physical HBA, then SMHBA2_HBA_ENA bit-wise ORed with the value of HBAOptions shall be one. If adapter virtualization is not enabled on this HBA, or if this is not a physical HBA, then SMHBA2_HBA_ENA bit-wise ORed with the value of HBAOptions shall be zero.

If this HBA is a virtual HBA (e.g., by use of PCI IOV), then SMHBA2_VHBA bit-wise ORed with the value of HBAOptions shall be one, SMHBA2_HBA_VA_SPT bit-wise ORed with the value of HBAOptions shall be zero, and SMHBA2_HBA_VA_USE bit-wise ORed with the value of HBAOptions shall be zero. If this HBA is not a virtual HBA, then SMHBA2_VHBA bit-wise ORed with the value of HBAOptions shall be zero.

All bits of the HBAOptions field not defined by this standard are reserved.

6.3.1.3.3 Manufacturer

Manufacturer shall be an ASCII string not exceeding 64 bytes the value of which is the name of the manufacturer of the HBA.

Example:

```
Hot Biscuits Adapters
```

6.3.1.3.4 SerialNumber

SerialNumber shall be an ASCII string not exceeding 64 bytes the value of which is the serial number of the HBA.

Example:

```
1040A-0000003
```

6.3.1.3.5 Model

Model shall be an ASCII string not exceeding 256 bytes the value of which is a vendor specific name or identifying text for the HBA product.

NOTE 2 - Some management applications limit the length of this attribute to 64 bytes.

Example:

```
HBA1040A
```

6.3.1.3.6 ModelDescription

ModelDescription shall be an ASCII string not exceeding 256 bytes the value of which is a description of the HBA product.

Example:

```
Hot Biscuits Adapters Short Form
```

6.3.1.3.7 HardwareVersion

HardwareVersion shall be an ASCII string not exceeding 256 bytes the value of which is a vendor specific identification of the hardware revision level of the HBA.

NOTE 3 - Some management applications limit the length of this attribute to 64 bytes.

6.3.1.3.8 DriverVersion

DriverVersion shall be an ASCII string not exceeding 256 bytes the value of which is a vendor specific identification of the driver version controlling this HBA.

6.3.1.3.9 OptionROMVersion

OptionROMVersion shall be an ASCII string not exceeding 256 bytes the value of which is a vendor specific identification of the option ROM or BIOS version version of the HBA.

6.3.1.3.10 FirmwareVersion

FirmwareVersion shall be an ASCII string not exceeding 256 bytes the value of which is a vendor specific identification of the firmware version of the HBA.

6.3.1.3.11 VendorSpecificID

VendorSpecificID shall have a vendor specific value.

6.3.1.3.12 DriverName

DriverName shall be an ASCII string not exceeding 256 bytes the value of which is the file name for the driver binary file. In the case of some operating systems that implement a generic driver name (e.g., Driver.o in UnixWare) an absolute path should be included in the driver name.

Example 1:

For NT 4.0 or Win2000 environment, it is the SCSI miniport driver name for the HBA (e.g., 1040AW2K.SYS is the name of the binary file for the SCSI miniport for the Hot Biscuits Adapters Short Form).

Example 2:

For UnixWare that uses generic driver name Driver.o, the full/absolute path should be used.

```
/etc/conf/pack.d/HotBiscuits/Driver.o
```

6.3.1.3.13 HBASymbolicName

HBASymbolicName shall be an ASCII string not exceeding 256 bytes the value of which is an identifying text for this HBA.

NOTE 4 - This attribute would be configured by the administrator through management applications.

6.3.1.3.14 RedundantOptionROMVersion

RedundantOptionROMVersion shall be an ASCII string not exceeding 256 bytes the value of which is a vendor specific identification of the redundant option ROM version of the HBA.

6.3.1.3.15 RedundantFirmwareVersion

RedundantFirmwareVersion shall be an ASCII string not exceeding 256 bytes the value of which is a vendor specific identification of the redundant firmware version of the HBA.

6.4 Adapter Bus Address

6.4.1 Generic Adapter Bus Address

6.4.1.1 Generic Adapter Bus Address Requirements

Requirements are specified in annex A for support of the attributes specified in 6.4.1.

6.4.1.2 Generic Adapter Bus Address Attribute Data Declarations

Any data object of type SMHBA_BUSTYPE shall have the format defined in this declaration:

```
typedef HBA_UINT8 SMHBA_BUSTYPE;
```

Any data object of type SMHBA_BUSTYPE shall have one of the values defined in this list:

```
#define SMHBA_PCI                          1      /* PCI adapter bus */
```

Any data object of type SMHBA2_BUSUNION shall have the format defined in this structure:

```
typedef union SMHBA2_BusUnion{
       SMHBA_PCIADDRESS     * PciAddress;
} SMHBA2_BUSUNION, *PSMHBA2_BUSUNION;
```

Any data object of type SMHBA2_BUSADDRESS shall have the format defined in this structure:

```
typedef struct SMHBA2_BusAddress {
       SMHBA_BUSTYPE        Type;
       SMHBA2_BUSUNION      Address;
} SMHBA2_BUSADDRESS, *PSMHBA2_BUSADDRESS;
```

6.4.1.3 Generic Adapter Bus Address Attribute Descriptions

6.4.1.3.1 Type

The Type field shall indicate the type of bus address. If the Type field is set to SMHBA_PCI, the bus address is a PCI 3.0 (see PCI 3.0) bus address (PCI 3.0 is the only allowed value in this standard).

6.4.1.3.2 Address

The Address field shall contain a pointer to a bus address structure of the type indicated by the Type field (PCI 3.0 is the only allowed value in this standard).

6.4.2 PCI Adapter Bus Address Attributes

6.4.2.1 PCI Adapter Bus Address Requirements

Requirements are specified in annex A for support of the attributes specified in 6.4.2.

6.4.2.2 PCI Adapter Bus Address Attribute Data Declarations

Any data object of type SMHBA_PCIADDRESS shall have the format defined in this structure:

```
typedef struct SMHBA_PCIAddress {
       HBA_UINT8            BusNumber;
       HBA_UINT8            DeviceNumber;
       HBA_UINT8            FunctionNumber;
} SMHBA_PCIADDRESS, *PSMHBA_PCIADDRESS;
```

6.4.2.3 PCI Adapter Bus Address Attribute Specification

6.4.2.3.1 BusNumber

The BusNumber field shall be set to the Bus Number field of the PCI 3.0 CONFIG_ADDRESS register for the device on which the adapter is implemented.

6.4.2.3.2 DeviceNumber

The DeviceNumber field shall be set to the Device Number field of the PCI 3.0 CONFIG_ADDRESS register for the device on which the adapter is implemented.

6.4.2.3.3 FunctionNumber

The FunctionNumber field shall be set to the Function Number field of the PCI 3.0 CONFIG_ADDRESS register for the device on which the adapter is implemented.

6.5 Port Attributes

6.5.1 Generic Port

6.5.1.1 Generic Port Requirements

Requirements are specified in annex A for support of the attributes specified in 6.5.1.

6.5.1.2 Generic Port Attribute Data Declarations

Any data object of type SMHBA2_PORTTYPE shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_PORTTYPE;
```

Any data object of type SMHBA2_PORTTYPE shall have one of the values defined in this list:

```
#define HBA_PORTTYPE_UNKNOWN              1      /* Unknown */
#define HBA_PORTTYPE_OTHER                2      /* Other */
#define SMHBA2_PORTTYPE_VNPORT            10     /* FC Virtual end port*/
#define SMHBA2_PORRTYPE_VFPORT            11     /* FC Virtual fabric port*/
#define HBA_PORTTYPE_SASDEVICE            30     /* SAS (SSP or STP) */
#define HBA_PORTTYPE_SATADEVICE           31     /* SATA Device, i.e. Direct Attach SATA */
#define HBA_PORTTYPE_SASEXPANDER          32     /* SAS Expander */
```

Any data object of type SMHBA2_PORTSTATE shall have the format defined in in this declaration:

```
typedef HBA_UINT32 SMHBA2_PORTSTATE;
```

Any data object of type SMHBA2_PORTSTATE shall have one of the values defined in this list:

```
#define SMHBA2_PORTSTATE_UNKNOWN          1      /* Unknown */
#define SMHBA2_PORTSTATE_ONLINE           2      /* Fully Operational */
#define SMHBA2_PORTSTATE_OFFLINE          3      /* Not provisioned or disabled */
```

Any data object of type SMHBA2_PORT shall have the format defined in this structure:

```
typedef union SMHBA2_Port{
       SMHBA2_FC_PORT       * FCPort;
       SMHBA2_SAS_PORT      * SASPort;
} SMHBA2_PORT, *PSMHBA2_PORT;
```

NOTE 5 - The attributes of a VN_Port or VF_Port are represented by an SMHBA2_FC_PORT structure, regardless of its associated link level medium.

Any data object of the type SMHBA2_PORTATTRIBUTES shall have the format defined in this structure:

```
typedef struct SMHBA2_PortAttributes {
       HBA_HANDLE           PortHandle;
       SMHBA2_PORTTYPE      PortType;
       SMHBA2_PORTSTATE     PortState;
       char                 OSDeviceName[256];
       SMHBA2_PORT          PortSpecificAttributes;
} SMHBA2_PORTATTRIBUTES, *PSMHBA2_PORTATTRIBUTES;
```

6.5.1.3 Generic Port Attribute Specifications

6.5.1.3.1 PortHandle

PortHandle is an integer that uniquely identifies the Port among all Ports within an instance of the HBA API software. The value of PortHandle shall not change once it is assigned (e.g., at power-up, or hot-plug of an Adapter), and shall not be reassigned to a different Port unless by reinitialization of the HBA API software.

6.5.1.3.2 PortType

PortType shall be an integer that indicates the port type of the SMHBA2_Port. It shall have a value defined in 6.5.1.2.

6.5.1.3.3 PortState

PortState shall be an integer that indicates the current state of the SMHBA2_Port. PortState shall be set to:

a) SMHBA2_PORTSTATE_UNKNOWN if the state of the port is unknown;
b) SMHBA2_PORTSTATE_ONLINE if the port is capable of communication with other ports; and
c) SMHBA2_PORTSTATE_OFFLINE if the port has no associated Phy, or has been administratively conditioned so that it does not pass data.

6.5.1.3.4 OSDeviceName

OSDeviceName shall be an ASCII string not exceeding 256 bytes the value of which is the device name that the SMHBA2_Port is visible from the operating system, if known. It shall be a zero length ASCII string if the SMHBA2_Port is not an FC end port or an SAS device end port.

If an OSDeviceName is provided by the SMHBA API in an SMHBA2_PortAttributes structure, it shall comply with these rules:

a) A non-null end port OSDeviceName shall be provided if it is possible to use that name in operating system specific functions to affect the same end port as is specified in the other fields in the rest of the structure;
b) If there are any names that have the preferred format as specified in table 1 and also meet the requirements of rule a), then one of them shall be provided. If there are more than one, one shall be chosen and consistently provided (i.e., multiple calls shall provide the same name);
c) If there are no names with the preferred format as specified in table 1 but there are names that meet the requirements of rule a), then one of them shall be provided. If there are more than one, one shall be consistently provided (i.e., multiple calls shall provide the same name); and
d) If no name meets the requirements of rule a), the OSDeviceName shall be a zero length ASCII string.

Table 1 — Preferred format for SMHBA2_PortAttributes OSDeviceName (part 1 of 2)

OS	Preferred format [a]	
	Local end port	**Discovered end port**
AIX	**/dev/fscsi***x* (for a Fibre Channel N_Port. There is no preferred form for a SAS Port)	(zero length string)
Linux	**/dev/***name*	(zero length string)
Solaris	**/devices/***localportname* (for a Fibre Channel N_Port. There is no preferred form for a SAS Port)	**cX:***pwwx* (the attachment point ID, pwwx is the N_Port_Name for a Fibre Channel N_Port. There is no preferred form for a SAS Port.)

[a] In end port name format samples, text appearing in **bold weight** shall appear in the shown position as it appears in the format sample. Text appearing in *normal weight italics* is a placeholder for similar text determined by the rules of the OS. Italicized lower case *x* represents any decimal number and may be more than one digit. Normal text in parentheses is descriptive, not format sample.

Table 1 — Preferred format for SMHBA2_PortAttributes OSDeviceName (part 2 of 2)

OS	Preferred format [a]	
	Local end port	**Discovered end port**
Windows	**\\.\Scsi*x*:**	(zero length string)
HP-UX	**/dev/*drivernamex*** (a Fibre Channel N-port where *drivername* is the HBA driver name and *x* is the HBA instance number e.g. /dev/td0 for instance 0 of a Fibre Channel HBA using the td driver).	(zero length string)
OpenVMS	**FG*a*0:** (a Fibre Channel HBA with port id *a*, limit is 26 devices) **PK*a*0:** (a SAS HBA with port id *a*, limited is 26 devices, shared with iSCSI)	(zero length string)
vSphere	**vmhba*x***	(zero length string)

[a] In end port name format samples, text appearing in **bold weight** shall appear in the shown position as it appears in the format sample. Text appearing in *normal weight italics* is a placeholder for similar text determined by the rules of the OS. Italicized lower case *x* represents any decimal number and may be more than one digit. Normal text in parentheses is descriptive, not format sample.

6.5.1.3.5 PortSpecificAttributes

PortSpecificAttributes shall be an attribute structure for the specific FC or SAS Port type. The FC specific attributes are defined in 6.5.2.2 and the SAS specific atttributes are defined in 6.5.3.2. A port in an FCoE Fabric is represented as an FC Port type.

6.5.2 FC Port

6.5.2.1 FC Port Requirements

Requirements are specified in annex A for support of the attributes specified in 6.5.2.

6.5.2.2 FC Port Attribute Data Declarations

Any data object of type HBA_COS shall have a value specified for a Name Server Class of Service object TYPEs object (see FC-GS-7).

```
typedef HBA_UINT32 HBA_COS; /* see Class of Service - Format in FC-GS-7 */
```

Any data object of type HBA_FC4TYPES shall have a value specified for a Name Server FC-4 TYPEs object (see FC-GS-7).

```
typedef struct HBA_fc4types {
       HBA_UINT8 bits[32];  /* See FC-4 TYPEs - Format in FC-GS-7 */
}
```

The attributes of a VN_Port or VF_Port shall be represented by an SMHBA2_FC_Port structure, regardless of its associated link level medium.

Any data object of type SMHBA2_FC_PORT shall have the format defined in this structure:

```
typedef struct SMHBA2_FC_Port {
       HBA_WWN              NodeWWN;
       HBA_WWN              PortWWN;
       HBA_UINT32           AddressIdentifier;
       HBA_COS              PortSupportedClassofService;
       HBA_FC4TYPES         PortSupportedFc4Types;
       HBA_FC4TYPES         PortActiveFc4Types;
       char                 PortSymbolicName[256];
       HBA_UINT32           NumberofDiscoveredPorts;
}SMHBA2_FC_PORT, *PSMHBA2_FC_PORT;
```

6.5.2.3 FC Port Attribute Specifications

6.5.2.3.1 NodeWWN

NodeWWN shall be the Node_Name associated with this SMHBA2_FC_Port.

6.5.2.3.2 PortWWN

PortWWN shall be the FC_Port Name_Identifier of this SMHBA2_FC_Port.

6.5.2.3.3 AddressIdentifier

AddressIdentifier shall be an unsigned integer. The value of AddressIdentifier shall be the current Fibre Channel address identifier of the SMHBA2_FC_Port. The high order byte of AddressIdentifier shall contain the first byte of the address identifier, and successively lower order bytes of AddressIdentifier shall contain successive bytes of the address identifier. The lowest order byte of AddressIdentifier shall be zero.

6.5.2.3.4 PortSupportedClassofService

PortSupportedClassofService shall identify the supported classes of service of this SMHBA2_FC_Port. It shall have a value as defined in FC-GS-7 for Class of Service - Format.

6.5.2.3.5 PortSupportedFc4Types

PortSupportedFc4Types shall identify the FC-4 types to which this FC_Port and its software may be configured. It shall be zero if the FC_Port is not an end port. It shall have a value as defined in FC-GS-7 for FC-4 TYPEs - Format.

Bytes zero through three of PortSupportedFc4Types shall be the first word of the FC-4 TYPEs structure defined by GS-4, and successively higher numbered four byte groups of PortSupportedFc4Types shall be successive words of the FC-GS-7 FC-4 TYPEs structure. For each word, the lowest numbered byte of the group of four bytes in PortSupportedFc4Types shall be the highest order byte of the word in the FC-GS-7 FC-4 TYPEs structure, and successively higher numbered bytes of the group of bytes in PortSupportedFc4Types shall be successively lower order bytes of the word in the FC-GS-7 FC-4 TYPEs structure.

A Port Identifier supporting SCSI FCP (TYPE is 08h), ISO/IEC 8802-2 LLC/SNAP (In-order) (TYPE is 04h) and Fibre Channel Services (TYPE is 20h) would register the following PortSupportedFc4Types value:

00 00 01 10 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00h

6.5.2.3.6 PortActiveFc4Types

PortActiveFc4Types shall identify the FC-4 types that this FC_Port is configured to support. It shall be set to zero if the FC_Port is not an end port. It shall have a value as defined in FC-GS-7 for FC-4 TYPEs - Format.

Bytes zero through three of PortActiveFc4Types shall be the first word of the FC-4 TYPEs structure defined by GS-4, and successively higher numbered four byte groups of PortActiveFc4Types shall be successive words of the FC-GS-7 FC-4 TYPEs structure. For each word, the lowest numbered byte of the group of four bytes in PortActiveFc4Types shall be the highest order byte of the word in the FC-GS-7 FC-4 TYPEs structure, and successively higher numbered bytes of the group of bytes in PortActiveFc4Types shall be successively lower order bytes of the word in the FC-GS-7 FC-4 TYPEs structure.

6.5.2.3.7 PortSymbolicName

PortSymbolicName shall be an ASCII string not exceeding 256 bytes the value of which is the General Services Port Symbolic Name (see FC-GS-7). In a Fabric, this shall be the same as the entry registered with the Name Server.

6.5.2.3.8 NumberofDiscoveredPorts

For a local end port, the value of NumberofDiscoveredPorts shall be the number of end ports, regardless of their FC-4 support, and Fx_Ports that are visible to that local end port. At a minimum, this shall be the number of end ports mapped to a local SCSI device. It may reflect any superset of that minimum, up to all of the end ports and Fx_Ports on the fabric. For discovered SMHBA2_FC_Ports this value shall be zero.

6.5.3 SAS Port

6.5.3.1 SAS Port Requirements

Requirements are specified in annex A for support of the attributes specified in 6.5.3.

6.5.3.2 SAS Port Attribute Data Declarations

Any data object of type HBA_SASPORTPROTOCOL shall have the format defined in this declaration:

```
typedef HBA_UINT32  HBA_SASPORTPROTOCOL;
```

Any data object of the type HBA_SASPORTPROTOCOL shall have a value of zero or the bit-wise OR of one or more values defined in this list:

```
#define HBA_SASPORTPROTOCOL_SSP           1      /* Serial SCSI Protocol Port */
#define HBA_SASPORTPROTOCOL_STP           2      /* Serial ATA Tunneling Protocol Port */
#define HBA_SASPORTPROTOCOL_SMP           4      /* Serial Management Protocol Port */
#define HBA_SASPORTPROTOCOL_SATA          8      /* SATA Device, Direct Attached */
                                                 /* or anywhere in the domain. */
```

Any data object of type SMHBA2_SAS_PORT shall have the format defined in this structure:

```
typedef struct SMHBA2_SAS_Port {
       HBA_SASPORTPROTOCOL         PortProtocol;
       HBA_WWN                     LocalSASAddress;
       HBA_WWN                     AttachedSASAddress;
       HBA_UINT32                  NumberofDiscoveredPorts;
SMHBA2_SAS_PORT, *PSMHBA2_SAS_PORT;
```

6.5.3.3 SAS Port Attribute Specifications

6.5.3.3.1 PortProtocol

PortProtocol shall contain a value that is defined in and indicates all the protocols supported by this SAS Port.

6.5.3.3.2 LocalSASAddress

LocalSASAddress shall be the Port_Identifier of this SAS Port. Its value shall conform to the format as defined in the SAS specification (see SPL).

6.5.3.3.3 AttachedSASAddress

AttachedSASAddress shall be the SAS address of the entity at the other end of the SAS link from this local SAS Port. Its value shall conform to the format as defined in the SAS specification (see SPL).

6.5.3.3.4 NumberofDiscoveredPorts

NumberofDiscoveredPorts shall contain a value equal to the number of end ports that are visible to the local SAS Port. The discovered ports may also include SMP Ports contained within SAS expander devices.

6.6 Phy Attributes

6.6.1 Generic Phy

6.6.1.1 Generic Phy Requirements

Requirements are specified in annex A for support of the attributes specified in 6.6.1.

6.6.1.2 Generic Phy Attributes Data Declarations

Any data object of type SMHBA2_PHYTYPE shall have the format defined by this declaration:

```
typedef HBA_UINT32 SMHBA2_PHYTYPE;
```

Any data object of type SMHBA2_PHYTYPE shall have one of the values defined in this list:

```
#define SMHBA2_PHYTYPE_UNKNOWN            1 /* Unknown */
#define SMHBA2_PHYTYPE_NOTPRESENT         2 /* Not present */
#define SMHBA2_PHYTYPE_FC                 10 /* FC physical layer */
#define SMHBA2_PHYTYPE_ENET               20 /* Ethernet physical layer */
#define SMHBA2_PHYTYPE_SAS                30 /* SAS/SATA physical layer */
```

Any data object of type SMHBA2_PHY shall have the format defined by this declaration:

```
typedef union SMHBA2_Phy{
       SMHBA2_FC_PHY        * FCPhy;
       SMHBA2_SAS_PHY       * SASPhy;
       SMHBA2_ENET_PHY      * ENETPhy;
} SMHBA2_PHY, *PSMHBA2_PHY;
```

Any data object of type SMHBA2_PHYATTRIBUTES shall have the format defined by this structure:

```
typedef struct SMHBA2_PhyAttributes{
       HBA_HANDLE           PhyHandle;
       SMHBA2_PHYTYPE       PhyType;
       SMHBA2_PHY           PhySpecificAttributes;
} SMHBA2_PHYATTRIBUTES, *PSMHBA2_PHYATTRIBUTES;
```

6.6.1.3 Generic Phy Attributes Specifications

6.6.1.3.1 PhyHandle

PhyHandle is an integer that uniquely identifies the Phy among all Phys within an instance of the HBA API software. The value of PhyHandle shall not change once it is assigned (e.g., at power-up, or hot-plug of an Adapter or a Phy), and shall not be reassigned to a different Phy unless by reinitialization of the HBA API software.

6.6.1.3.2 PhyType

PhyType shall be set to the value that best represents the link technology for the Phy. If a Phy has been assigned a PhyIndex value and is partially or completely removed, its Phy-Type shall be SMHBA2_PHYTYPE_NOTPRESENT.

6.6.1.3.3 PhySpecificAttributes

PhySpecificAttributes shall be a pointer to an attribute structure for the type of Phy indicated by PhyType, as specified in table 2

Table 2 — Type of attributes structure pointer in PhyAttributes

Value of PhyType	Attribute pointer type
SMHBA2_PHYTYPE_UNKNOWN	null pointer
SMHBA2_PHYTYPE_NOTPRESENT	null pointer
SMHBA2_PHYTYPE_FC	PSMHBA2_FC_PHY (see 6.6.2)
SMHBA2_PHYTYPE_ENET	PSMHBA2_ENET_PHY (see 6.6.4)
SMHBA2_PHYTYPE_SAS	PSMHBA2_SAS_PHY (see 6.6.3)

6.6.2 FC Phy

6.6.2.1 FC Phy Requirements

Requirements are specified in annex A for support of the attributes specified in 6.6.2.

6.6.2.2 FC Phy Attribute Data Declaration

Any data object of type SMHBA2_FCPHYOPTIONS shall have the format defined in in this declaration:

```
typedef HBA_UINT32 SMHBA2_FCPHYOPTIONS;
```

Any data object of type SMHBA2_FCPHYOPTIONS shall have a value equal to zero or the bit-wise OR of one or more values defined in this list:

```
#define SMHBA2_FCPHYOPTIONS_TNG_SPT1      /* Phy is able and configured to support */
                                          /* Transmitter Training */
#define SMHBA2_FCPHYOPTIONS_TNG_RQ(1<<1)  /* Phy is configured to request */
                                          /* Transmitter Training */
#define SMHBA2_FCPHYOPTIONS_FEC_SPT(1<<2) /* Phy is able and configured to support */
                                          /* Forward Error Correction */
#define SMHBA2_FCPHYOPTIONS_FEC_RQ(1<<3)  /* Phy is configured to request */
                                          /* Forward Error Correction */
#define SMHBA2_FCPHYOPTIONS_FEC_ON(1<<4)  /* Phy is operating using */
                                          /* Forward Error Correction */
```

Any data object of type SMHBA2_FCPHYSPEED shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_FCPHYSPEED;
```

Any data object of type SMHBA2_FCPHYSPEED shall have a value equal to zero or the bit-wise OR of one or more values defined in this list:

```
#define SMHBA2_FCPHYSPEED_UNKNOWN         0            /* Unknown - */
                                                       /* transceiver incapable of
                                                       reporting */
#define SMHBA2_FCPHYSPEED_1GBIT           1            /* 1 GBit/sec */
#define SMHBA2_FCPHYSPEED_2GBIT           (1<<1)       /* 2 GBit/sec */
#define SMHBA2_FCPHYSPEED_4GBIT           (1<<2)       /* 4 GBit/sec */
#define SMHBA2_FCPHYSPEED_8GBIT           (1<<3)       /* 8 GBit/sec */
#define SMHBA2_FCPHYSPEED_16GBIT          (1<<4)       /* 16 GBit/sec */
#define SMHBA2_FCPHYSPEED_32GBIT          (1<<5)       /* 32 GBit/sec */
#define SMHBA2_FCPHYSPEED_64GBIT          (1<<6)       /* 64 GBit/sec */
#define SMHBA2_FCPHYSPEED_128GBIT         (1<<7)       /* 128 GBit/sec */
#define SMHBA2_FCPHYSPEED_10GBIT          (1<<14)      /* 10 GBit/sec */
#define SMHBA2_FCPHYSPEED_NOT_NEGOTIATED  (1<<15)      /* Speed not established */
```

Any data object of type SMHBA2_FCPHYSTATE shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_FCPHYSTATE;
```

Any data object of type SMHBA2_FCPHYSTATE shall have one of the values defined in this list:

```
#define SMHBA2_FCPHYSTATE_UNKNOWN         1      /* Unknown */
#define SMHBA2_FCPHYSTATE_ONLINE          2      /* Fully Operational */
#define SMHBA2_FCPHYSTATE_OFFLINE         3      /* Administratively Offline */
#define SMHBA2_FCPHYSTATE_BYPASSED        4      /* Bypassed */
#define SMHBA2_FCPHYSTATE_DIAGNOSTICS     5      /* In diagnostics mode */
#define SMHBA2_FCPHYSTATE_LINKDOWN        6      /* Link Down */
#define SMHBA2_FCPHYSTATE_ERROR           7      /* Phy Error */
#define SMHBA2_FCPHYSTATE_LOOPBACK        8      /* Loopback */
#define SMHBA2_FCPHYSTATE_DEGRADED        9      /* Degraded, but Operational mode */
```

Any data object of type SMHBA2_FCPHYTOPOLOGY shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_FCPHYTOPOLOGY;
```

Any data object of type SMHBA2_FCPHYTOPOLOGY shall have one of the values defined in this list:

```
#define SMHBA2_FCPHYTOPOLOGY_UNKNOWN      1      /* Unknown */
#define SMHBA2_FCPHYTOPOLOGY_PTP          10     /* point-to-point link */
#define SMHBA2_FCPHYTOPOLOGY_LOOP         11     /* private loop */
#define SMHBA2_FCPHYTOPOLOGY_FABRIC       12     /* point-to-point link to a Fabric */
#define SMHBA2_FCPHYTOPOLOGY_LFABRIC      13     /* loop to a Fabric */
```

Any data object of type SMHBA2_FCMEDIATYPE shall have the format defined in this declaration:

```
typedef HBA_UINT8 SMHBA2_FCMEDIATYPE;
```

Any data object of type SMHBA2_FCMEDIATYPE shall have one of the values defined in this list:

```
#define SMHBA2_FCMEDIATYPE_UNKNOWN        1      /* Unknown Phy type */
#define SMHBA2_FCMEDIATYPE_OPTICAL        2      /* Optical Phy */
#define SMHBA2_FCMEDIATYPE_COPPER         4      /* Copper Phy */
```

A data object of type SMHBA2_FC_PHY shall only be associated with a Phy that has SMHBA2_PHYTYPE (see 6.6.1.2) set to SMHBA2_PHYTYPE_FC.

Any data object of type SMHBA2_FC_PHY shall have the format defined in this structure:

```
typedef struct SMHBA2_FC_Phy {
       SMHBA2_FCPHYOPTIONS         PhyOptions;
       SMHBA2_FCPHYSPEED           PhySupportedSpeed;
       SMHBA2_FCPHYSPEED           PhySpeed;
       SMHBA2_FCPHYSTATE           PhyState;
       SMHBA2_FCPHYTOPOLOGY        PhyTopology;
       SMHBA2_FCMEDIATYPE          MediaType;
       HBA_UINT32                  MaxFrameSize;
} SMHBA2_FC_PHY, *PSMHBA2_FC_PHY;
```

6.6.2.3 FC Phy Attribute Specifications

6.6.2.3.1 PhyOptions

PhyOptions shall be an integer that indicates the current settings of FC-specific optional behaviors.

If Transmitter Training is supported and adminstratively allowed for this FC Phy, then SMHBA2_FCPHYOPTIONS_TNG_SPT bit-wise ORed with the value of PhyOptions shall be one. If Transmitter Training is not supported or not administratively allowed on this FC Phy, then SMHBA2_FCPHYOPTIONS_TNG_SPT bit-wise ORed with the value of PhyOptions shall be zero. The current operating speed of the Phy is not relevant to this setting.

If Transmitter Training is configured to be requested by this FC Phy, then SMHBA2_FCPHYOPTIONS_TNG_RQ bit-wise ORed with the value of PhyOptions shall be one. If Transmitter Training is not configured to be requested by this FC Phy, then SMHBA2_FCPHYOPTIONS_TNG_RQ bit-wise ORed with the value of PhyOptions shall be zero. The current operating speed of the Phy is not relevant to this setting.

If Forward Error Correction is supported for this FC Phy, then SMHBA2_FCPHYOPTIONS_FEC_SPT bit-wise ORed with the value of PhyOptions shall be one. If Forward Error Correction is not supported on this FC Phy, then SMHBA2_FCPHYOPTIONS_FEC_SPT bit-wise ORed with the value of PhyOptions shall be zero. The current operating speed of the Phy is not relevant to this setting.

If Forward Error Correction is supported, and configured to be requested by this FC Phy, then SMHBA2_FCPHYOPTIONS_FEC_RQ bit-wise ORed with the value of PhyOptions shall be one. If Forward Error Correction is not supported or configured to be requested by this FC Phy, then SMHBA2_FCPHYOPTIONS_FEC_RQ bit-wise ORed with the value of PhyOptions shall be zero. The current operating speed of the Phy is not relevant to this setting.

If Forward Error Correction is currently in use by this FC Phy, then SMHBA2_FCPHYOPTIONS_FEC_ON bit-wise ORed with the value of PhyOptions shall be one. If Forward Error Correction is not currently in use by this FC Phy, then SMHBA2_FCPHYOPTIONS_FEC_ON bit-wise ORed with the value of PhyOptions shall be zero.

All bits of the PhyOptions field not defined by this standard are reserved.

6.6.2.3.2 PhySupportedSpeed

PhySupportedSpeed shall identify all the signalling bit rates at which FC Phy may operate. It shall have a value that is defined in 6.6.2.2. It may identify multiple speeds.

6.6.2.3.3 PhySpeed

PhySpeed shall identify the signalling rate at which the FC Phy is currently operating. It shall have a value that is defined in 6.6.2.2. It shall indicate only a single speed.

6.6.2.3.4 PhyState

PhyState shall be an integer that indicated the current operational state of the FC Phy. It shall have a value that is defined in 6.6.2.2.

6.6.2.3.5 PhyProtocol

PhyTopology shall be an integer that indicates the topology in which the FC Phy is operating. It shall have a value that is defined in 6.6.2.2.

6.6.2.3.6 MediaType

MediaType shall identify the type of the physical transport media supported by the FC Phy. It shall have a value that is defined in 6.6.2.2.

6.6.2.3.7 MaxFrameSize

MaxFrameSize shall contain a value equal to the maximum frame size in bytes supported by this FC Phy.

6.6.3 SAS Phy

6.6.3.1 SAS Phy requirements

Requirements are specified in annex A for support of the attributes specified in 6.6.3.

6.6.3.2 SAS Phy Attribute Data Declaration

Any object of type SMHBA2_SASPHYSPEED shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_SASPHYSPEED;
```

Any object of type SMHBA2_SASPHYSPEED shall have one of the values defined in this list:

```
#define HBA_SASSTATE_UNKNOWN              0x00   /* Phy is enabled. Speed is unknown */
#define HBA_SASSTATE_DISABLED             0x01   /* Phy is disabled. */
#define HBA_SASSTATE_FAILED               0x02   /* Phy is enabled. But failed speed */
                                                 /* negotiation. */
#define HBA_SASSTATE_SATASPINUP           0x03   /* Phy is enabled. */
                                                 /* Detected a SATA device and */
                                                 /* entered the SATA Spinup hold state */
#define HBA_SASSTATE_SATAPORTSEL          0x04   /* The phy is attached to */
                                                 /* a Port Selector (see SATA-2.6). */
#define HBA_SASSTATE_RESET_IN_PROGRESS    0x05   /* Phy is enabled. */
                                                 /* Expander is performing */
                                                 /* SMP PHY CONTROL. */
#define HBA_SASSTATE_PHY_UNSUPPORTED      0x06   /* Phy is enabled. */
```

```
                                                /* Unsupported settings detected. */
#define HBA_SASSPEED_1_5GBIT              0x08   /* Phy enabled at 1.5 GBit/sec */
#define HBA_SASSPEED_3GBIT                0x09   /* Phy enabled at 3 GBit/sec */
#define HBA_SASSPEED_6GBIT                0x0A   /* Phy enabled at 6 GBit/sec */
#define SMHBA2_SASSPEED_12GBIT            0x0B   /* Phy enabled at 12 Gbit/sec */
```

A data object of type SMHBA2_SAS_PHY shall only be associated with a Phy that has SMHBA2_PHYTYPE (see 6.6.1.2) set to SMHBA2_PHYTYPE_SAS.

Any data object of type SMHBA2_SAS_PHY shall have the format defined in this structure:

```
typedef struct SMHBA2_SAS_Phy {
       HBA_UINT8                   PhyIdentifier;
       SMHBA2_SASPHYSPEED          NegotiatedLinkRate;
       SMHBA2_SASPHYSPEED          ProgrammedMinLinkRate;
       SMHBA2_SASPHYSPEED          HardwareMinLinkRate;
       SMHBA2_SASPHYSPEED          ProgrammedMaxLinkRate;
       SMHBA2_SASPHYSPEED          HardwareMaxLinkRate;
       HBA_WWN                     domainPortWWN;
} SMHBA2_SAS_PHY, *PSMHBA2_SAS_PHY;
```

6.6.3.3 SAS Phy Attribute Specifications

6.6.3.3.1 PhyIdentifier

PhyIdentifier shall uniquely identify the Phy whose physical configuration and link information is being returned. It is unique within the context of the SAS device that contains the Phy. It shall have a value as defined in SPL Phy Identifiers.

6.6.3.3.2 NegotiatedLinkRate

NegotiatedLinkRate shall identify the state or the transmission speed negotiated by the Phy for the physical link. It shall have a value defined in 6.6.3.2.

6.6.3.3.3 ProgrammedMinLinkRate

ProgrammedMinLinkRate indicates the minimum physical link rate set by the Phy control mechanism. It shall have a value defined in 6.6.3.2. A value of zero indicates that the minimum physical link rate is not programmable.

6.6.3.3.4 HardwareMinLinkRate

HardwareMinLinkRate indicates the minimum physical link rate supported by the Phy. It shall have a value defined in 6.6.3.2.

6.6.3.3.5 ProgrammedMaxLinkRate

ProgrammedMaxLinkRate indicates the maximum physical link rate set by the Phy control mechanism. It shall have a value defined in 6.6.3.2. A value of zero indicates that maximum physical link rate is not programmable.

6.6.3.3.6 HardwareMaxLinkRate

HardwareMaxLinkRate indicates the maximum physical link rate supported by the Phy. It shall have a value defined in 6.6.3.2.

6.6.3.3.7 domainPortWWN

domainPortWWN shall be the Port_Identifier with the smallest value of any Port_Identifier of an expander SMP port discovered through the phy. It shall have a value of 0 if no SMP port has been discovered through the phy.

6.6.4 Ethernet Phy

6.6.4.1 Ethernet Phy requirements

Requirements are specified in annex A for support of the attributes specified in 6.6.4.

6.6.4.2 Ethernet Phy Attributes Data Declarations

Any data object of type SMHBA2_ENETPHYOPTIONS shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_ENETPHYOPTIONS;
```

Any data object of type SMHBA2_ENETPHYOPTIONS shall have a value equal to zero.

Any data object of type SMHBA2_ENETPHYSPEED shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_ENETPHYSPEED;
```

Any data object of type SMHBA2_ENETPHYSPEED shall have a value equal to zero or the bit-wise OR of one or more values defined in this list:

```
#define SMHBA2_ENETSPEED_UNKNOWN          0      /* Unknown - */
                                                 /* transceiver incapable of reporting */
#define SMHBA2_ENETPHYSPEED_1GBIT         1      /* 1 GBit/sec */
#define SMHBA2_ENETPHYSPEED_10GBIT        (1<<1) /* 10 GBit/sec */
#define SMHBA2_ENETPHYSPEED_40GBIT        (1<<2) /* 40 GBit/sec */
#define SMHBA2_ENETPHYSPEED_100GBIT       (1<<3) /* 100 GBit/sec */
#define SMHBA2_ENETPHYSPEED_NOT_NEGOTIATED(1<<31)/* Speed not established yet*/
```

Any data object of type SMHBA2_ENPHYSTATE shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_ENPHYSTATE;
```

Any data object of type SMHBA2_ENPHYSTATE shall have one of the values defined in this list:

```
#define SMHBA2_ENPHYSTATE_UNKNOWN         1      /* Unknown */
#define SMHBA2_ENPHYSTATE_ONLINE          2      /* Fully Operational */
#define SMHBA2_ENPHYSTATE_OFFLINE         3      /* Administratively Offline */
#define SMHBA2_ENPHYSTATE_DIAGNOSTICS     4      /* In diagnostics mode */
#define SMHBA2_ENPHYSTATE_LINKDOWN        5      /* Link Down */
#define SMHBA2_ENPHYSTATE_ERROR           6      /* Phy Error */
#define SMHBA2_ENPHYSTATE_LOOPBACK        7      /* Loopback */
#define SMHBA2_ENPHYSTATE_DEGRADED        8      /* Degraded, but Operational mode */
```

Any data object of type SMHBA2_ENETMEDIATYPE shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_ENETMEDIATYPE;
```

Any data object of type SMHBA2_ENETMEDIATYPE shall have one of the values defined in this list:

```
#define SMHBA2_ENETMEDIATYPE_UNKNOWN      1      /* Unknown Phy type */
#define SMHBA2_ENETMEDIATYPE_OPTICAL      2      /* Optical Phy */
#define SMHBA2_ENETMEDIATYPE_COPPER       3      /* Copper Phy */
#define SMHBA2_ENETMEDIATYPE_WIRELESS     4      /* Wireless Phy */
```

A data object of type SMHBA2_ENET_PHY shall only be associated with a PHY that has SMHBA2_PHYTYPE (see 6.6.1.2) set to SMHBA2_PHYTYPE_ENET.

Any data object of type SMHBA2_ENET_PHY shall have the format defined in this structure:

```
typedef struct SMHBA2_Enet_Phy {
       SMHBA2_ENETPHYOPTIONS       PhyOptions;
       SMHBA2_ENETPHYSPEED         PhySupportedSpeed;
       SMHBA2_ENETPHYSPEED         PhySpeed;
       SMHBA2_ENPHYSTATE           PhyState;
       SMHBA2_ENETMEDIATYPE        MediaType;
       HBA_UINT32                  MaxFrameSize;
       HBA_UINT32                  VLANMask[128];
} SMHBA2_ENET_PHY, *PSMHBA2_ENET_PHY;
```

6.6.4.3 Ethernet Phy Attributes Specifications

6.6.4.3.1 PhyOptions

PhyOptions shall be set to zero.

6.6.4.3.2 PhySupportedSpeed

PhySupportedSpeed shall identify all the signalling bit rates at which the Ethernet Phy may operate. It shall have a value that is defined in 6.6.4.2. It may identify multiple speeds.

6.6.4.3.3 PhySpeed

PhySpeed shall identify the signalling rate at which the Ethernet Phy is currently operating. It shall have a value that is defined in 6.6.4.2. It shall indicate only a single speed.

6.6.4.3.4 PhyState

PhyState shall be an integer that indicates the current operational state of the Ethernet Phy. It shall have a value that is defined in 6.6.4.2.

6.6.4.3.5 MediaType

MediaType shall identify the type of the physical transport media supported by the Ethernet Phy. It shall have a value that is defined in 6.6.4.2.

6.6.4.3.6 MaxFrameSize

MaxFrameSize shall be set to the maximum Ethernet frame size in bytes supported by this Ethernet Phy. This may be larger than the size necessary to encapsulate the maximum FCoE frame size.

6.6.4.3.7 VLANMask

If the Ethernet Phy is not using VLAN tagging (see IEEE 802.1Q-2005), then the value of VLANMask shall be zero.

If the Ethernet Phy is using VLAN tagging, then the value of VLANMask shall be a bit mask constructed by setting to one the bits of VLANMask at the numerical offsets of each of the VLAN tag accessible through this Ethernet Phy, and setting to zero all other bits of VLANMask. This value may be zero if the Ethernet Phy controller has not completed initiailazation and discovery. Numerical offsets of bits in VLANMask shall be determined as follows:

a) the least significant bit of VLANMask(0) has offset 0;

b) for all n such that 0<n<128, the least significant bit of VLANMask(n) has offset 32 greater than the offset of VLANMask(n-1); and
c) for any bit other than the least significant bit in any VLANMask(n), the offset of that bit is one greater than the offset of the next less significant bit.

6.7 N_Port Controller

6.7.1 Generic N_Port Controller

6.7.1.1 Generic N_Port Controller Requirements

Requirements are specified in annex A for support of the attributes specified in 6.7.1.

6.7.1.2 Generic N_Port Controller Attributes Data Declarations

Any data object of type SMHBA2_NPCTYPE shall have the format defined by this declaration:

```
typedef HBA_UINT32 SMHBA2_NPCTYPE;
```

Any data object of type SMHBA2_NPCTYPE shall have one of the values defined in this list:

```
#define SMHBA2_NPCTYPE_FC          10     /* FC physical layer */
#define SMHBA2_NPCTYPE_FCOE        20     /* Ethernet physical layer */
```

Any data object of type SMHBA2_NPCLTR shall have the format defined by this declaration:

```
typedef union SMHBA2_NPCtlr{
       SMHBA2_FC_NPC        * FCNPCtlr;
       SMHBA2_FCOE_CTLR     * FCoECtlr;
} SMHBA2_NPCLTR, *PSMHBA2_NPCLTR;
```

Any data object of type SMHBA2_NPCATTRIBUTES shall have the format defined by this structure:

```
typedef struct SMHBA2_NPCAttributes{
       HBA_HANDLE                 NPCHandle;
       SMHBA2_NPCTYPE             NPCType;
       SMHBA2_NPCTLR              NPCSpecificAttributes;
} SMHBA2_NPCATTRIBUTES, *PSMHBA2_NPCATTRIBUTES;
```

6.7.1.3 N_Port Controller Attributes Specifications

6.7.1.3.1 NPCHandle

NPCHandle is an integer that uniquely identifies the N_Port Controller among all N_Port Controllers within an instance of the HBA API software. The value of NPCHandle shall not change once it is assigned (e.g., at power-up, or hot-plug of an Adapter or a Phy), and shall not be reassigned to a different N_Port Controller unless by reinitialization of the HBA API software.

6.7.1.3.2 NPCType

NPCType shall be set to the value that represents the link technology for the Phy associated with the N_Port Controller. It shall have a value that is defined in 6.7.1.2.

6.7.1.3.3 NPCSpecificAttributes

NPCSpecificAttributes shall be a pointer to an attribute structure for the type of N_Port Controller indicated by NPCType, as specified in table 3.

Table 3 — Type of attributes structure pointer in NPCAttributes

Value of PhyType	Attribute pointer type
SMHBA2_NPCTYPE_FC	PSMHBA2_FC_NPC (see 6.7.2)
SMHBA2_NPCTYPE_FCOE	PSMHBA2_FCOE_CTLR (see 6.7.3)

6.7.2 FC N_Port Controller

6.7.2.1 FC N_Port Controller requirements

Requirements are specified in annex A for support of the attributes specified in 6.7.2.

6.7.2.2 FC N_Port Controller Attributes Data Declarations

Any data object of type SMHBA2_NPCOPTIONS shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_NPCOPTIONS;
```

Any data object of type SMHBA2_NPCOPTIONS shall have a value that is zero or the bit-wise OR of one or more values defined in this list:

```
#define SMHBA2_NPCVF_SPT    (1<<0)        /* Fabric virtualization supported */
#define SMHBA2_NPCVF_ENA    (1<<1)        /* Fabric virtualization enabled */
#define SMHBA2_NPCVP_SPT    (1<<2)        /* N_Port_ID virtualization supported */
#define SMHBA2_NPCVP_ENA    (1<<3)        /* N_Port_ID virtualization enabled */
```

A data object of type SMHBA2_FC_NPC shall only be associated with an N_Port Controller that has NPCType (see 6.7.1.2) set to SMHBA2_NPCTYPE_FC.

Any data object of type SMHBA2_FC_NPC shall have the format defined in this structure:

```
typedef struct SMHBA2_FC_NPC {
       SMHBA2_NPCOPTIONS    FCNOptions;
       HBA_UINT32           VFIDMask[128];
       HBA_WWN              CoreNPortName;
       HBA_WWN              CoreSwitchName;
       HBA_UINT32           PortVfid;
} SMHBA2_FC_NPC, *PSMHBA2_FC_NPC;
```

6.7.2.3 FC N_Port Controller Attributes Specifications

6.7.2.3.1 FCNOptions

If Virtual Fabric Tagging is supported and adminstratively allowed for this N_Port Controller, then SMHBA2_NPCVF_SPT bit-wise ORed with the value of FCNOptions shall be one. If Virtual Fabric Tagging is not supported or not administratively allowed on this N_Port Controller, then SMHBA2_NPCVF_SPT bit-wise ORed with the value of FCNOptions shall be zero.

If Virtual Fabric Tagging is enabled for use between this N_Port Controller and its VF_Port, then SMHBA2_NPCVF_ENA bit-wise ORed with the value of FCNOptions shall be one. If Virtual Fabric

Tagging is not enabled for use between this N_Port Controller and its VF_Port, then SMHBA2_NPCVF_ENA bit-wise ORed with the value of FCNOptions shall be zero.

If N_Port virtualization (i.e., NPIV) is supported and adminstratively allowed for this N_Port Controller and will be requested from the Fabric, then SMHBA2_NPCVP_SPT bit-wise ORed with the value of FCNOptions shall be one. If N_Port virtualization (i.e., NPIV) is not supported or not administratively allowed on this N_Port Controller, then SMHBA2_NPCVP_SPT bit-wise ORed with the value of FCNOptions shall be zero.

If N_Port virtualization (i.e., NPIV) is enabled for use between this N_Port Controller and its VF_Port, then SMHBA2_NPCVP_ENA bit-wise ORed with the value of FCNOptions shall be one. If N_Port virtualization (i.e., NPIV) is not enabled for use between this N_Port Controller and its VF_Port, then SMHBA2_NPCVP_ENA bit-wise ORed with the value of FCNOptions shall be zero.

All bits of the FCNOptions field not defined by this standard are reserved.

6.7.2.3.2 VFIDMask

If SMHBA2_NPCVF_ENA bit-wise ORed with the value of FCNOptions is zero, then the value of VFIDMask shall be zero.

If SMHBA2_NPCVF_ENA bit-wise ORed with the value of FCNOptions is one, then the value of VFIDMask shall be a bit mask constructed by setting to one the bits of VFIDMask at the numerical offsets of each of the Virtual Fabric IDs accessible through this N_Port Controller and setting to zero all other bits of VFIDMask. This value may be zero if the N_Port Controller has not completed the EVFT protocol (see FC-LS-2). Numerical offsets of bits in VFIDMask shall be determined by:

a) the least significant bit of VFIDMask(0) has offset 0;
b) for all n such that 0<n<128, the least significant bit of VFIDMask(n) has offset 32 greater than the offset of VFIDMask(n-1); and
c) for any bit other than the least significant bit in any VFIDMask(n), the offset of that bit is one greater than the offset of the next less significant bit.

6.7.2.3.3 CoreNPortName

If the N_Port Controller does not enable Virtual Fabric Tagging, then the value of CoreNPortName shall be zero.

If this N_Port Controller enables Virtual Fabric Tagging, then the value of CoreNPortName shall be the CoreNPortName value configured for this N_Port Controller.

6.7.2.3.4 CoreSwitchName

CoreSwitchName shall be determined by:

a) if the N_Port Controller does not enable Virtual Fabric Tagging, then the value of CoreSwitchName shall be zero;
b) if this N_Port Controller enables Virtual Fabric Tagging and has not determined to use VF tagging via the the EVFP protocol (see FC-LS-2), then the value of CoreSwitchName shall be zero; or
c) if this N_Port Controller has determined to use VF tagging via the the EVFP protocol, then the value of CoreSwitchName shall be the value for CoreSwitchName most recently returned from the Fabric in an EVFT_Sync ELS or an EVFT_SYNC LS_ACC ELS via this N_Port Controller.

6.7.2.3.5 PortVfid

PortVfid shall be determined by:

a) if the N_Port Controller does not enable Virtual Fabric Tagging, then the value of PortVfid shall be zero;
b) if this N_Port Controller enables Virtual Fabric Tagging and has not determined to use VF tagging via the the EVFP protocol (see FC-LS-2), then the value of PortVfid shall be the Port VF_ID value configured for this N_Port Controller, which may be zero; or
c) if this N_Port Controller enables Virtual Fabric Tagging and has determined to use VF tagging via the the EVFP protocol then the value of PortVfid shall be the value for Port VF_ID most recently sent to the Fabric in an EVFT_SYNC ELS or an EVFT_SYNC LS_ACC ELS via this N_Port Controller.

6.7.3 ENode FCoE Controller

6.7.3.1 Overview

The operations of one FCoE Controller of an ENode MAC (see FC-BB-6) are performed on one VLAN (i.e., there is one SMHBA2_FCOE_CTLR data structure for each instance of an FCoE Controller).

6.7.3.2 ENode FCoE Controller requirements

Requirements are specified in annex A for support of the attributes specified in 6.7.3.

6.7.3.3 ENode FCoE Controller Attributes Data Declarations

Any data object of type SMHBA2_FCOECTLROPTIONS shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_FCOECTLROPTIONS;
```

Any data object of type SMHBA2_FCOECTLROPTIONS shall have a value that is zero or the bit-wise OR of one or more values defined in this list:

```
#define SMHBA2_FCOECTLR_VP_SPT (1<<0) /* N_Port_ID virtualization supported */
#define SMHBA2_FCOECTLR_VP_ENA (1<<1) /* N_Port_ID virtualization enabled */
```

Any data object of type SMHBA2_DiscoveredFCFMACAddr shall have the format defined in this structure:

```
typedef struct SMHBA2_DiscoveredFCFMACAddr {
       SMHBA2_MACADDR       FCFMACAddr;
       HBA_UINT8            MaxFCoESizeVerifiedBit;
       HBA_UINT8            AvailableForLoginBit;
} SMHBA2_DISCOVEREDFCFMACADDR, *PSMHBA2_DISCOVEREDFCFMACADDR;
```

Any data object of type SMHBA2_DiscoveredFCFMACAddrList shall have the format defined in this structure:

```
typedef struct SMHBA2_DiscoveredFCFMACAddrList {
       HBA_UINT32    NumberOfEntries;
       HBA_HANDLE    DiscoveredFCFMACAddr[1];
} SMHBA2_DISCOVEREDFCFMACADDRLIST, *PSMHBA2_DISCOVEREDFCFMACADDRLIST;/* A list of
Discovered FCF-MAC Addresses */
```

A data object of type SMHBA2_FCOE_CTLR shall only be associated with an N_Port Controller that has NPCType (see 6.7.1.2) set to SMHBA2_NPCTYPE_FCOE.

Any data object of type SMHBA2_FCOE_CTLR shall have the format defined in this structure:

```
typedef struct SMHBA2_FCoE_Ctlr {
       SMHBA2_FCOECTLROPTIONS           FCoECtlrOptions;
       SMHBA2_MACADDR                   ENodeMAC;
       HBA_UINT32                       FCoECtlrVLTag;
       SMHBA2_DISCOVEREDFCFMACADDRLIST  *DiscoveredFCFMACAddrList;
```

```
} SMHBA2_FCOE_CTLR, *PSMHBA2_FCOE_CTLR;
```

6.7.3.4 ENode FCoE Controller Attributes Specifications

6.7.3.4.1 FCoECtlrOptions

If N_Port virtualization (i.e., NPIV) is supported and adminstratively allowed for this FCoE Controller and will be requested from the Fabric, then SMHBA2_FCOECTLR_VP_SPT bit-wise ORed with the value of FCoECtlrOptions shall be one. If N_Port virtualization (i.e., NPIV) is not supported or not administratively allowed on this FCoE Controller, then SMHBA2_FCOECTLR_VP_SPT bit-wise ORed with the value of FCoECtlrOptions shall be zero.

If N_Port virtualization (i.e., NPIV) is enabled for use between this FCoE Controller and its VF_Port, then SMHBA2_FCOECTLR_VP_ENA bit-wise ORed with the value of FCoECtlrOptions shall be one. If N_Port virtualization (i.e., NPIV) is not enabled for use between this FCoE Controller and its VF_Port, then SMHBA2_FCOECTLR_VP_ENA bit-wise ORed with the value of FCoECtlrOptions shall be zero.

All bits of the FCoECtlrOptions field not defined by this standard are reserved.

6.7.3.4.2 ENode MAC

ENodeMAC shall be the MAC of the ENode MAC address associated with this FCoE Controller.

6.7.3.4.3 FCoECtlrVLTag

FCoECtlrVLTag shall be the VLAN tag value associated with this FCoE Controller.

6.7.3.4.4 FCFMACAddr

FCFMACAddr shall be the value of one FCF-MAC in the list of FCF-MACs created by the FCoE Controller of an ENode MAC during ENode/FCF Discovery (see FC-BB-6).

6.7.3.4.5 MaxFCoESizeVerifiedBit

MaxFCoESizeVerifiedBit shall be the value of the ‘Max FCoE Size Verified’ bit for FCFMACAddr (see FC-BB-6).

6.7.3.4.6 AvailableForLoginBit

AvailableForLoginBit shall be the value of the ‘AvailableForLogin’ bit for FCFMACAddr (see FC-BB-6).

6.7.4 FCoE Link Endpoint

6.7.4.1 FCoE Link Endpoint requirements

Requirements are specified in annex A for support of the attributes specified in 6.7.4.

6.7.4.2 FCoE Link Endpoint Attributes Data Declarations

Any data object of type SMHBA2_FCoE_LEP shall have the format defined in this structure:

```
typedef struct SMHBA2_FCoE_LEP {
       SMHBA2_MACADDR       LEPVNPortMAC;
       SMHBA2_MACADDR       LEPFCFMAC;
       HBA_UINT32           LEPVLTag;
       HBA_UINT8            BeaconPeriod;
```

```
       HBA_UINT8            FKAADVPeriod;
} SMHBA2_FCOE_LEP, *PSMHBA2_FCOE_LEP;
```

6.7.4.3 ENode Link Endpoint Specifications

6.7.4.3.1 LEPVNPortMAC

LEPVNPortMAC shall be the VN_Port MAC that this Link Endpoint uses (see FC-BB-6).

6.7.4.3.2 LEPFCFMAC

LEPFCFMAC shall be the FCF-MAC that this Link Endpoint uses (see FC-BB-6).

6.7.4.3.3 LEPVLTag

LEPVLTag shall be the VLAN tag value that this VN_Port/FCoE_LEP pair uses (see FC-BB-6).

6.7.4.3.4 BeaconPeriod

BeaconPerdiod shall contain a value equal to the Locally Unique N_Port_ID timer BEACON_PERIOD, in milliseconds, for VN2VN Enode MACs (see FC-BB-6).

6.7.4.3.5 FKAADVPeriod

FDAADVPeriod shall contain a value equal to the advertised FKA_ADV_PERIOD, in milliseconds (see FC-BB-6).

6.8 Fabric Attributes

6.8.1 Fabric Info

6.8.1.1 Fabric Info requirements

Requirements are specified in annex A for support of the attributes specified in 6.8.1

6.8.1.2 Fabric Info Data Declarations

Any data object of type SMHBA2_FABRICFLAGS shall have the format defined in this declaration:

```
typedef HBA_UINT32 SMHBA2_FABRICFLAGS;
```

Any data object of type SMHBA2_FABRICFLAGS shall have a value that is zero or the inclusive OR of zero or more values defined in this list:

```
#define SMHBA_BROADCAST           (1<<3)       /* Broadcast routing  supported*/
#define SMHBA_FCSP                (1<<5)       /* FC-SP Authentication supported*/
#define SMHBA_CLASS2              (1<<7)       /* Class 2 Service  supported*/
#define SMHBA_CLASS3              (1<<8)       /* Class 3 Service  supported*/
#define SMHBA_INORDER             (1<<12)      /* Sequential delivery in Class 3
                                                  guaranteed*/
```

Any data object of type SMHBA2_FABRICINFO shall represent a single Fabric and shall have the format defined in this structure:

```
typedef struct SMHBA2_FabricInfo {
       HBA_HANDLE                  FabricHandle;
       HBA_WWN                     FabricName;
       SMHBA2_FABRICFLAGS          Flags;
       HBA_UINT32                  Ratov;
       HBA_UINT32                  Edtov;
} SMHBA2_FABRICINFO, *PSMHBA2_FABRICINFO;
```

6.8.1.3 Fabric Info Specifications

6.8.1.3.1 FabricHandle

FabricHandle is an integer that uniquely identifies the Fabric among all Fabrics within an instance of the HBA API software. The value of FabricHandle shall not change once it is assigned (e.g., at power-up, or hot-plug of an Adapter), and shall not be reassigned to a different Fabric unless by reinitialization of the HBA API software.

6.8.1.3.2 FabricName

FabricName shall contain the FabricName value returned from the Fabric in an FLOGI or FDISC LS_ACC (see FC-LS-2).

6.8.1.3.3 Flags

Each defined bit in the Flags field is set to a value constructed to represent the capabilities defined in 6.8.1.2 most recently returned from the Fabric in an FLOGI LS_ACC.

All bits of the Flags field not defined by this standard are reserved.

6.8.1.3.4 Ratov

Ratov shall contain:the R_A_TOV value most recently returned from the Fabric in an FLOGI LS_ACC.

6.8.1.3.5 Edtov

Edtov shall contain:the E_D_TOV value most recently returned from the Fabric in an FLOGI LS_ACC.

6.9 Statistics

6.9.1 Protocol Statistics

6.9.1.1 Protocol Statistics requirements

Requirements are specified in annex A for support of the attributes specified in 6.9.1.

Unless otherwise specified, statistics counters shall be 64-bit signed integers that shall wrap to zero after reaching their maximum value. They shall not be changed other than to count the events they represent.

> NOTE 6 - Since statistics counters are not changed other than to count the events they represent, event rates may be determined by the difference of time and counter values at two successive calls, with appropriate algorithms to deal with counter wrap.

6.9.1.2 Protocol Statistics Data Declarations

Any data object of type SMHBA_PROTOCOLSTATISTICS shall represent activity for a single Upper Level Protocol and shall have the format defined in this structure:

```
typedef struct SMHBA_ProtocolStatistics {
       HBA_INT64            SecondsSinceLastReset;
       HBA_INT64            InputRequests;
       HBA_INT64            OutputRequests;
       HBA_INT64            ControlRequests;
       HBA_INT64            InputMegabytes;
       HBA_INT64            OutputMegabytes;
} SMHBA_PROTOCOLSTATISTICS, *PSMHBA_PROTOCOLSTATISTICS;
```

6.9.1.3 Protocol Statistics Attribute Specifications

6.9.1.3.1 SecondsSinceLastReset

SecondsSinceLastReset shall contain a value equal to the number of seconds since the statistics were last reset.

6.9.1.3.2 InputRequests

InputRequests shall contain a value equal to the number of protocol specific operations (e.g., FC-4, SMP) causing data input. Some single protocol requests may cause both input and output of data. If these requests occur, they shall be counted in both InputRequests and OutputRequests. The sum of InputRequests and OutputRequests may exceed the total number of requests.

6.9.1.3.3 OutputRequests

OutputRequests shall contain a value equal to the number of protocol specific operations (e.g., SSP) causing protocol specific data output. Some single protocol requests may cause both input and output of data. If these requests occur, they shall be counted in both InputRequests and OutputRequests. The sum of InputRequests and OutputRequests may exceed the total number of requests.

6.9.1.3.4 ControlRequests

ControlRequests shall contain a value equal to the number of protocol specific operations (e.g., FC-4) that do not permit data movement.

6.9.1.3.5 InputMegabytes

InputMegabytes shall contain a value equal to the number of megabytes (see 3.1.43) of protocol specific (e.g., STP) data input.

6.9.1.3.6 OutputMegabytes

OutputMegabytes shall contain a value equal to the number of megabytes (see 3.1.43) of protocol specific (e.g., STP) data output.

6.9.2 Port Statistics

6.9.2.1 Port Statistics requirements

Requirements are specified in annex A for support of the attributes specified in 6.9.2. Unless otherwise specified, statistics counters shall be 64-bit signed integers that shall wrap to zero after reaching their maximum value. They shall not be changed other than to count the events they represent.

NOTE 7 - Since statistics counters are not changed other than to count the events they represent, event rates may be determined by the difference of time and counter values at two successive calls, with appropriate algorithms to deal with counter wrap.

6.9.2.2 Port Statistics Data Declarations

Any data object of type SMHBA_PORTSTATISTICS shall represent activity for a single VN_Port and shall have the format defined in this structure:

```
typedef struct SMHBA_PortStatistics {
       HBA_INT64            SecondsSinceLastReset;
       HBA_INT64            TxFrames;
       HBA_INT64            TxWords;
       HBA_INT64            RxFrames;
       HBA_INT64            RxWords;
}SMHBA_PORTSTATISTICS, *PSMHBA_PORTSTATISTICS;
```

6.9.2.3 Port Statistics Descriptions

6.9.2.3.1 SecondsSinceLastReset

SecondsSinceLastReset shall contain a value equal to the number of seconds since the statistics were last reset for the port with which this statistics structure is associated.

6.9.2.3.2 TxFrames

TxFrames shall contain a value equal to the number of total Transmitted Fibre Channel or SAS frames across all protocols for the port with which this statistics structure is associated.

6.9.2.3.3 RxFrames

RxFrames shall contain a value equal to the number of total Received Fibre Channel frames across all protocols for the port with which this statistics structure is associated.

6.9.2.3.4 TxWords

TxWords shall contain a value equal to the number of total Transmitted Fibre Channel words across all protocols for the port with which this statistics structure is associated.

6.9.2.3.5 RxWords

RxWords shall contain a value equal to the number of total Received Fibre Channel words across all protocols for the port with which this statistics structure is associated.

6.9.3 Phy Statistics

6.9.3.1 Phy Statistics Data Declaration

Any data object of type SMHBA2_PHYSTATISTICS shall represent activity for a single Phy and shall have the format defined in this declaration:

```
typedef union SMHBA2_PhyStatistics {
       SMHBA_SASPHYSTATISTICS      * SASPhyStatistics;
       SMHBA2_FCPHYSTATISTICS      * FCPhyStatistics;
       SMHBA2_ENPHYSTATISTICS      * ENPhyStatistics;
} SMHBA2_PHYSTATISTICS, *PSMHBA2_PHYSTATISTICS;
```

6.9.4 SAS Phy Statistics

6.9.4.1 SAS Phy Statistics requirements

Requirements are specified in annex A for support of the attributes specified in 6.9.4.

Unless otherwise specified, statistics counters shall be 64-bit signed integers that shall wrap to zero after reaching their maximum value. They shall not be changed other than to count the events they represent.

NOTE 8 - Since statistics counters are not changed other than to count the events they represent, event rates may be determined by the difference of time and counter values at two successive calls, with appropriate algorithms to deal with counter wrap.

6.9.4.2 SAS Phy Statistics Data Declaration

```
typedef struct SMHBA_SASPhyStatistics {
       HBA_INT64            SecondsSinceLastReset;
       HBA_INT64            TxFrames;
       HBA_INT64            TxWords;
       HBA_INT64            RxFrames;
       HBA_INT64            RxWords;
       HBA_INT64            InvalidDwordCount;
       HBA_INT64            RunningDisparityErrorCount;
       HBA_INT64            LossofDwordSyncCount;
       HBA_INT64            PhyResetProblemCount;
} SMHBA_SASPHYSTATISTICS, *PSMHBA_SASPHYSTATISTICS;
```

6.9.4.3 SAS Phy Statistics Attribute Specifications

6.9.4.3.1 SecondsSinceLastReset

SecondsSinceLastReset shall contain a value equal to the number of seconds since the statistics were last reset.

6.9.4.3.2 TxFrames

TxFrames shall contain a value equal to the number of the total transmitted Serial Attached SCSI frames across all protocols.

6.9.4.3.3 TxWords

TxFrames shall contain a value equal to the number of the total transmitted SAS dwords across all protocols.

6.9.4.3.4 RxFrames

RxFrames shall contain a value equal to the number of the total received Serial Attached SCSI frames across all protocols.

6.9.4.3.5 RxWords

RxWords shall contain a value equal to the number of the total received SAS dwords across all protocols.

6.9.4.3.6 InvalidDwordCount

InvalidDwordCount shall contain a value equal to the value of the INVALID DWORD COUNT field of the SAS phy log descriptor for the specified phy (see SPL). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

NOTE 9 - This field is only defined for a phy associated with an SSP target port.

6.9.4.3.7 RunningDisparityErrorCount

RunningDisparityErrorCount shall contain a value equal to the value of the RUNNING DISPARITY ERROR COUNT field of the SAS phy log descriptor for the specified phy (see SPL). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

NOTE 10 - This field is only defined for a phy associated with an SSP target port.

6.9.4.3.8 LossofDwordSynchronizationCount

LossofDwordSynchronizationCount shall contain a value equal to the value of the LOSS OF DWORD SYNCHRONIZATION field of the SAS phy log descriptor for the specified phy (see SPL). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

NOTE 11 - This field is only defined for a phy associated with an SSP target port.

6.9.4.3.9 PhyResetProblemCount

PhyResetProblemCount shall contain a value equal to the value of the PHY RESET PROBLEM COUNT field of the SAS phy log descriptor for the specified phy (see SPL). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

NOTE 12 - This field is only defined for a phy associated with an SSP target port.

6.9.5 FC Phy Statistics

6.9.5.1 FC Phy Statistics requirements

Requirements are specified in annex A for support of the attributes specified in 6.9.5.

Unless otherwise specified, statistics counters shall be 64-bit signed integers that shall wrap to zero after reaching their maximum value. They shall not be changed other than to count the events they represent.

NOTE 13 - Since statistics counters are not changed other than to count the events they represent, event rates may be determined by the difference of time and counter values at two successive calls, with appropriate algorithms to deal with counter wrap.

6.9.5.2 FC Phy Statistics Data Declaration

```
/* Statistical counters for FC-0, FC-1, and FC-2 */

typedef struct SMHBA2_FCPhyStatistics {
       HBA_INT64            SecondsSinceLastReset;
       HBA_INT64            TxFrames;
       HBA_INT64            TxWords;
       HBA_INT64            RxFrames;
       HBA_INT64            RxWords;
       HBA_INT64            LIPCount;
       HBA_INT64            NOSCount;
       HBA_INT64            ErrorFrames;
       HBA_INT64            DumpedFrames;
       HBA_INT64            LinkFailureCount;
       HBA_INT64            LossOfSyncCount;
       HBA_INT64            LossOfSignalCount;
       HBA_INT64            PrimitiveSeqProtocolErrCount;
       HBA_INT64            InvalidTxWordCount;
       HBA_INT64            InvalidCRCCount;
       HBA_INT64            FLOGICount;
       HBA_INT64            FLOGOCount;
}SMHBA2_FCPHYSTATISTICS, *PSMHBA2_FCPHYSTATISTICS;
```

6.9.5.3 FC Phy Statistics Attribute Specifications

6.9.5.3.1 SecondsSinceLastReset

SecondsSinceLastReset shall contain a value equal to the number of seconds since the statistics were last reset.

6.9.5.3.2 TxFrames

TxFrames shall contain a value equal to the number of total Transmitted Fibre Channel frames across all protocols and classes.

6.9.5.3.3 RxFrames

RxFrames shall contain a value equal to the number of total Received Fibre Channel frames across all protocols and classes.

6.9.5.3.4 TxWords

TxWords shall contain a value equal to the number of total Transmitted Fibre Channel words across all protocols and classes.

6.9.5.3.5 RxWords

RxWords shall contain a value equal to the number of total Received Fibre Channel words across all protocols and classes.

6.9.5.3.6 LIPCount

LIPCount shall contain a value equal to the number of LIP Primitive Sequences that have occurred on an Arbitrated Loop.

6.9.5.3.7 NOSCount

NOSCount shall contain a value equal to the number of NOS Primitive Sequences that have occurred on the switched Fabric.

6.9.5.3.8 ErrorFrames

ErrorFrames shall contain a value equal to the number of frames that have been received in error.

6.9.5.3.9 DumpedFrames

DumpedFrames shall contain a value equal to the number of frames that were lost due to a lack of host buffers available.

6.9.5.3.10 LinkFailureCount

LinkFailureCount shall contain a value equal to the value of the Link Failure Count field of the Link Error Status Block for the specified end port (see FC-FS-3). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

6.9.5.3.11 LossOfSyncCount

LossOfSyncCount shall contain a value equal to the value of the Loss-of-Synchronization Count field of the Link Error Status Block for the specified end port (see FC-FS-3). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

6.9.5.3.12 LossOfSignalCount

LossOfSignalCount shall contain a value equal to the value of the Loss-of-Signal Count field of the Link Error Status Block for the specified end port (see FC-FS-3). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

6.9.5.3.13 PrimitiveSeqProtocolErrCount

PrimitiveSeqProtocolErrCount shall contain a value equal to the value of the Primitive Sequence Protocol Error field of the Link Error Status Block for the specified end port (see FC-FS-3). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

6.9.5.3.14 InvalidTxWordCount

InvalidTxWordCount shall contain a value equal to the value of the Invalid transmission Word field of the Link Error Status Block for the specified end port (see FC-FS-3). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

6.9.5.3.15 Invalid CRC Count

InvalidCRCCount shall contain a value equal to the value of the Invalid CRC Count field of the Link Error Status Block for the specified end port (see FC-FS-3). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

6.9.5.3.16 FLOGICount

FLOGICount shall contain a value equal to the number of Fabric LOGI requests transmitted.

6.9.5.3.17 FLOGOCount

FLOGOCount shall contain a value equal to the number of Fabric LOGO requests transmitted.

6.9.6 Ethernet Phy Statistics

6.9.6.1 Ethernet Phy Statistics Compliance

Requirements are specified in annex A for support of the attributes specified in 6.9.6.

Unless otherwise specified, statistics counters shall be 64-bit signed integers that shall wrap to zero after reaching their maximum value. They shall not be changed other than to count the events they represent.

NOTE 14 - Since statistics counters are not changed other than to count the events they represent, event rates may be determined by the difference of time and counter values at two successive calls, with appropriate algorithms to deal with counter wrap.

6.9.6.2 Ethernet Phy Statistics Data Declaration

```
typedef struct SMHBA2_ENPhyStatistics {
      HBA_INT64              SecondsSinceLastReset;
      HBA_INT64              TxENFrames;
      HBA_INT64              TxENBytes;
      HBA_INT64              RxENFrames;
      HBA_INT64              RxENBytes;
      HBA_INT64              LinkFailureCount;
      HBA_INT64              SymbolErrorCount;
      HBA_INT64              ErroredBlockCount;
      HBA_INT64              FCSErrorCount;
}SMHBA2_ENPHYSTATISTICS, *PSMHBA2_ENPHYSTATISTICS;
```

6.9.6.3 Ethernet Phy Statistics Attribute Specifications

6.9.6.3.1 SecondsSinceLastReset

SecondsSinceLastReset shall contain a value equal to the number of seconds since the statistics were last reset.

6.9.6.3.2 TxENFrames

TxENFrames shall contain a value equal to the number of total Transmitted Ethernet frames across all protocols.

6.9.6.3.3 TxENBytes

TxENBytes shall contain a value equal to the number of total Transmitted Ethernet bytes across all protocols.

6.9.6.3.4 RxENFrames

RxENFrames shall contain a value equal to the number of total Received Ethernet frames across all protocols.

6.9.6.3.5 RxENBytes

RxENBytes shall contain a value equal to the number of total Received Ethernet bytes across all protocols.

6.9.6.3.6 LinkFailureCount

LinkFailureCount shall contain a value equal to the value of the Link Failure Count field of the Link Error Status Block for the specified Ethernet port (see FC-BB-6). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

6.9.6.3.7 SymbolErrorCount

SymbolErrorCount shall contain a value equal to the value of the Symbol Error During Carrier Count field of the Link Error Status Block for the specified Ethernet port (see FC-BB-6). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

6.9.6.3.8 ErroredBlockCount

ErroredBlockCount shall contain a value equal to the value of the Errored Block Count field of the Link Error Status Block for the specified Ethernet port (see FC-BB-6). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

6.9.6.3.9 FCSErrorCount

FCSErrorCount shall contain a value equal to the value of the Frame Check Sequence Error Count field of the Link Error Status Block for the specified Ethernet port (see FC-BB-6). It is a 64-bit signed integer that does not increment past the maximum value representable by a 32-bit unsigned integer.

6.9.7 FIP Statistics

6.9.7.1 FIP Statistics requirements

Requirements are specified in annex A for support of the attributes specified in 6.9.7. Unless otherwise specified, statistics counters shall be 64-bit signed integers that shall wrap to zero after reaching their maximum value. They shall not be changed other than to count the events they represent.

NOTE 15 - Since statistics counters are not changed other than to count the events they represent, event rates may be determined by the difference of time and counter values at two successive calls, with appropriate algorithms to deal with counter wrap.

6.9.7.2 FIP Statistics Data Declarations

```
typedef struct SMHBA2_FIPStatistics {
       HBA_INT64 SecondsSinceLastReset;
       HBA_INT64 FIPVLANNotifications;
       HBA_INT64 FCFTimeoutCount;
       HBA_INT64 BEACONTimeoutCount;
       HBA_INT64 FIPMulticastAdvertReceivedCount;
       HBA_INT64 KeepAliveSentCount;
       HBA_INT64 ClearVirtualLinksReceivedCount;
}SMHBA2_FIPSTATISTICS, *PSMHBA2_FIPSTATISTICS;
```

6.9.7.3 FIP Statistics Descriptions

6.9.7.3.1 SecondsSinceLastReset

SecondsSinceLastReset shall contain a value equal to the number of seconds since the statistics were last reset.

6.9.7.3.2 FIPVLANNotifications

FIPVLANNotifications shall contain a value equal to the number of VLAN notifications received on all VLANs (see FC-BB-6).

6.9.7.3.3 FCFTimeoutCount

FCFTimeoutCount shall contain a value equal to the number of times a link has been deinstantiated due to FCF timeout (i.e., no unsolicited multicast Discovery Advertisement received by the FCoE Controller of the ENode MAC within 2.5 x FKA_ADV_PERIOD (see FC-BB-6).

6.9.7.3.4 BEACONTimeoutCount

BEACONTimeoutCount shall contain a value equal to the number of times a link has been deinstantiated due to a Beacon timeout (i.e., N_Port_ID Beacon or N_Port_ID P2P Beacon has been received within 2.5 x BEACON_PERIOD (see FC-BB-6).

6.9.7.3.5 FIPMulticastAdvertReceivedCount

FIPMulticastAdvertReceivedCount shall contain a value equal to the number of multicast Discovery Advertisements received by the FCoE Controller of the ENode MAC (see FC-BB-6).

6.9.7.3.6 KeepAliveSentCount

KeepAliveSentCount shall contain a value equal to the number of FIP Keep Alive frames sent by the FCoE Controller of the ENode MAC (see FC-BB-6).

6.9.7.3.7 ClearVirtualLinksReceivedCount

ClearVirtualLinksReceivedCount shall contain a value equal to the number FIP Clear Virtual Links frames received by the FCoE Controller of the ENode MAC (see FC-BB-6).

6.10 Target Port Attributes

6.10.1 Target Port Attribute Data Declaration

6.10.1.1 SMHBA_BIND_CAPABILITY

Any data object of type SMHBA_BIND_CAPABILITY shall have a value defined in this subclause.

```
typedef HBA_UINT32 SMHBA_BIND_CAPABILITY;

#define  SMHBA_CAN_BIND_TO_WWPN 0x0001
#define  SMHBA_CAN_BIND_TO_LUID 0x0002
#define  SMHBA_CAN_BIND_ANY_LUNS 0x0400
#define  SMHBA_CAN_BIND_AUTOMAP 0x0800
```

6.10.1.2 SMHBA_BIND_TYPE

Any data object of type SMHBA_BIND_TYPE shall have a value defined in this subclause.

```
typedef HBA_UINT32 SMHBA_BIND_TYPE;

#define SMHBA_BIND_TO_WWPN 0x0001
#define SMHBA_BIND_TO_LUID 0x0002
```

6.10.1.3 SMHBA_ScsiId

```
typedef struct SMHBA_ScsiId {
       char                OSDeviceName[256];
       HBA_UINT32          ScsiBusNumber;
       HBA_UINT32          ScsiTargetNumber;
       HBA_UINT32          ScsiOSLun;
} SMHBA_SCSIID, *PSMHBA_SCSIID;
```

6.10.1.4 SMHBA_LUID

```
typedef struct SMHBA_LUID {
       char          buffer[256];
} SMHBA_LUID, *PSMHBA_LUID;
```

6.10.1.5 SMHBA_PORTLUN

```
typedef struct SMHBA_PORTLUN {
       HBA_WWN             PortWWN;
       HBA_WWN             domainPortWWN;
       SMHBA_SCSILUN       TargetLun;
} SMHBA_PORTLUN, *PSMHBA_PORTLUN;
```

6.10.1.6 Composite types

```
typedef struct SMHBA_ScsiEntry {
       SMHBA_SCSIID ScsiId;
       SMHBA_PORTLUN PortLun;
       SMHBA_LUID LUID;
} SMHBA_SCSIENTRY, *PSMHBA_SCSIENTRY;

typedef struct SMHBA_TargetMapping {
       HBA_UINT32 NumberOfEntries;
       SMHBA_SCSIENTRY entry[1];          /* Variable length array containing */
                                          /* mappings */
} SMHBA_TARGETMAPPING, *PSMHBA_TARGETMAPPING;

typedef struct SMHBA_BindingEntry {
       SMHBA_BIND_TYPE      type;
       SMHBA_SCSIID         ScsiId;
       SMHBA_PORTLUN        PortLun;
       SMHBA_LUID           LUID;
       HBA_STATUS           Status;
} SMHBA_BINDINGENTRY, *PSMHBA_BINDINGENTRY;

typedef struct SMHBA_Binding {
       HBA_UINT32                  NumberOfEntries;
       SMHBA_BINDINGENTRY          entry[1]; /* Variable length array */
} SMHBA_BINDING, *PSMHBA_BINDING;
```

6.10.2 Target Mapping and Persistent Binding Attribute Specifications

6.10.2.1 Overview

A target mapping is a pairing of an SMHBA_SCSIID attribute and an SMHBA_PORTLUN attribute. It represents a relationship currently in effect such that SCSI operations requested by applications with respect to the OS device represented by the SMHBA_SCSIID attribute act on the SCSI logical unit identified by the SMHBA_PORTLUN attribute. Each HBA is presumed to provide a list of one or more target mappings to the OS via its driver. The collection of all target mappings by all HBAs is the OS view of its SAN resources. More than one mapping for any one OS device identifier (i.e., SMHBA_SCSIID attribute) is not allowed, though more than one OS device identifier may map to the same SCSI logical unit (i.e., SMHBA_PORTLUN attribute). This standard specifies no requirements for how target mappings are established, though they may be affected by persistent bindings.

A persistent binding is a pairing of an SMHBA_SCSIID attribute and an SMHBA_PORTLUN attribute that is retained through reinitialization of the OS, HBA, and/or fabric, and/or domain, and establishes a target mapping subsequent to reinitialization. This standard specifies but does not require means to establish and remove persistent bindings (see 7.7). An HBA may also use vendor specific means to establish and remove persistent bindings.

6.10.2.2 SMHBA_BIND_CAPABILITY

A data object of type SMHBA_BIND_CAPABILITY shall represent support for a specific set of features related to persistent binding. Each HBA end port together with its driver software has certain implemented persistent binding capabilities. Additionally, an HBA end port together with its driver software may allow the availability of some persistent binding capabilities it implements to be enabled or disabled. Any data object of type SMHBA_BIND_CAPABILITY shall have a value equal to the bit-wise OR of one or more symbolic constants declared in 6.10.1.1 and defined in 6.10.3.

6.10.2.3 SMHBA_BIND_TYPE

A data object of type SMHBA_BIND_TYPE shall indicate a set of persistent binding features that qualify a specific persistent binding. Any data object of type SMHBA_BIND_TYPE shall have a value equal to the bit-wise OR of one or more symbolic constants declared in 6.10.1.2 and defined in 6.10.3.

6.10.2.4 SMHBA_SCSIID

A data object of type SMHBA_SCSIID shall encapsulate an operating system identification of a SCSI logical unit. The value of its OSDeviceName field shall be as specified in 6.10.2.9. The value of its ScsiBusNumber field shall be as specified in 6.10.2.10. The value of its ScsiTargetNumber field shall be as specified in 6.10.2.11. The value of its ScsiOSLun field shall be as specified in 6.10.2.12.

> NOTE 16 - Many versions of Windows and Unix and their application programs identify storage resources via an abstraction of the SCSI Parallel Interface architecture (see SPI-5) (i.e., a resource is identified by the OS as though it is a SCSI logical unit within a SCSI target device accessed by a SCSI controller). The means of identification is a numeric triplet comprising controller or bus number, target number, and logical unit number (LUN). This may in turn be further abstracted to a device in the OS file system, and thereby identified by its device name, as a character string.

6.10.2.5 SMHBA_LUID

A data object of type SMHBA_LUID shall contain a value equal to an Identification Descriptor from the Vital Products Data Device Identification Page (i.e., VPD Page 83h, see SPC-4) returned by a logical unit in reply to a SCSI INQUIRY command as specified in SPC-4 with these additional constraints:

a) its length shall be 256 bytes or less;
b) its Association value shall be logical unit association (0h); and
c) its Identifier Type shall be one of:
   A) Vendor Specific (0h);
   B) T10 vendor identification (1h);
   C) EUI-64 (2h); or
   D) Name_Identifier as defined in FC-FS-3 (3h).

An Identification Descriptor of Identifier Type 2h or 3h should be used if the related logical unit provides any Identification Descriptor of these Identifier Types. A vendor specific LUID has no assurance of uniqueness or persistence. A vendor specific LUID should be used only if it is the only alternative, or its persistence and uniqueness are known by the local administration to be sufficient.

6.10.2.6 PortWWN

Within the context of any FCP-4 target port or SSP target port attribute data structures defined in 6.10.1, the PortWWN field shall contain one of the following values:

a) Zero; or
b) N_Port_Name of an FCP-4 target port; or
c) Port_Identifier of an SSP target port;

Within the context of a target mapping returned from an SM-HBA-2 API function, the PortWWN field shall contain the N_Port_Name of the FCP-4 target port or Port_Identifier of an SSP target port that is represented in the mapping.

NOTE 17 - Since SAS specification does not define Target port Names, the SM-HBA-2 API shall use the Port_Identifier of the Target port as defined by SAS specification as PortWWN.

6.10.2.7 domainPortWWN

Within the context of any FCP-4 target port attribute data structures defined in 6.10.1, the domainPortWWN field shall be ignored.

Within the context of any SSP target port attribute data structures defined in 6.10.1, if the structure is returned by a function, the domainPortWWN field shall contain one of the following values:

a) If there are no discovered expanders, zero; or
b) If there are discovered expanders, the Port_Identifier with the smallest value of any Port_Identifier of an expander SMP port discovered in the same SAS domain as the SSP target port identified by the Port_Identifier in the same structure.

Within the context of any SSP target port attribute data structures defined in 6.10.1, if the structure is passed into a function, the domainPortWWN field shall contain one of the following values:

a) Zero; or
b) Port_Identifier of any expander SMP target port that has been discovered in the same SAS domain as the SSP target port identified by the Port_Identifier in the same structure.

If the domainPortWWN field passed into a function contains zero and the SSP target port that is represented in the same structure identifies more than one discovered SSP target port, the function shall indicate a status of HBA_STATUS_ERROR_AMBIGUOUS_WWN.

NOTE 18 - Since SAS specification does not define Target port names, the SM-HBA-2 API shall use the Port_Identifier of a discovered expander SMP port as defined by SAS specification as domainPortWWN.

6.10.2.8 TargetLun

Within the context of any FCP-4 target port or SSP target port attribute data structures defined in 6.10.1, the TargetLun field shall contain zero or the 64-bit SCSI LUN of a SCSI logical unit (see SAM-5) accessed through the FCP-4 target port or SSP target port.

Within the context of a target mapping returned from an SM-HBA-2 API function, the TargetLun field shall contain the 64-bit SCSI LUN of the logical unit that is mapped.

6.10.2.9 OSDeviceName

Within the context of FCP-4 target port or SSP target port attribute data structures defined in 6.10.1, the OSDeviceName field shall contain an ASCII string that is null or the name by which the operating system represents a SCSI logical unit (see SAM-5) to application programs. This attribute is an ASCII string with length from 1 to 256 bytes.

If an OSDeviceName is provided by the SM-HBA-2 API in an SMHBA_SCSIID structure within an SMHBA_TARGETMAPPING structure, it shall comply with the following rules:

a) A non-null logical unit OSDeviceName shall be provided if it is possible to use that name in operating system specific functions to affect the same logical unit as is referenced by the other fields in the rest of the structure;
b) If there are any names that have the preferred format as specified in table 4 and also comply with rule a), then one of them shall be provided. If there are more than one, one shall be chosen and consistently provided (i.e., multiple calls shall provide the same name);
c) If there are no names with the preferred format as specified in table 4 but there are names that comply with rule a), then one of them shall be provided. If there are more than one, one shall be chosen and consistently provided (i.e., multiple calls shall provide the same name); and
d) If no name complies with rule a), the OSDeviceName shall be a zero length ASCII string.

Table 4 — Preferred format for logical unit OSDeviceName (part 1 of 4)

OS	Preferred format for logical unit type [a]			
	disk/optical	**cd-rom**	**tape**	**changer**
AIX	**/dev/hdisk***x* (disk) (or) **/dev/omd***x* (optical)	**/dev/cd***x*	**/dev/rmt***x*	(zero length string)
Linux	**/dev/sd***a*	**/dev/scd***x*	**/dev/st***x*	(zero length string)
Solaris	**/dev/rdsk/c***x***t***y***d***z***s2** [b]	**/dev/rdsk/c***x***t***y***d***z***s2**	**/dev/rmt/***x***n**	(zero length string)
Windows	**\\.\PHYSICALDRIVE***x*	**\\.\CDROM***x*	**\\.\TAPE***x*	**\\.\CHANGER***x*

[a] In logical unit name format samples, text appearing in **bold weight** shall appear in the shown position as it appears in the format sample. Text appearing in *normal weight italics* is a placeholder for similar text determined by the rules of the OS. Italicized lower case *x* or *y* or *z* represents any decimal number and may be more than one digit. Italicized lower case *a* represents one or two lower case alphabetic characters. Italicized lower case *b* represents the null terminated string created from the designator/identifier field associated with the logical unit in the Device Identification VPD page (see SPC-4). Normal text in parentheses is descriptive, not format sample.

[b] These names shall reference the raw (i.e., unformatted) and unpartitioned disk. So long as it is consistent with relevant Solaris OS documentation, rdsk shall be used to indicate the raw device and s2 shall be used to reference the unpartitioned disk. Should other formats be established by relevant OS documentation for representing these characteristics, the new formats shall also be considered preferred formats.

Table 4 — Preferred format for logical unit OSDeviceName (part 2 of 4)

OS	Preferred format for logical unit type [a]			
	disk/optical	**cd-rom**	**tape**	**changer**
HP-UX For versions 11i V1 &11i V2	**/dev/dsk/c*x*t*y*d*z*** (block device. The "*x*" in the **c*x*t*y*d*z*** device files is the instance number of the controller of the bus that the device is on) **/dev/rdsk/c*x*t*y*d*z*** (character device. The "*x*" in the **c*x*t*y*d*z*** device files is the instance number of the controller of the bus that the device is on.)	**/dev/dsk/c*x*t*y*d*z*** (block device. The "*x*" in the **c*x*t*y*d*z*** device files is the instance number of the controller of the bus that the device is on) **/dev/rdsk/c*x*t*y*d*z*** (character device. The "*x*" in the **c*x*t*y*d*z*** device files is the instance number of the controller of the bus that the device is on.)	**/dev/mt/*x*m*options*** (or) **/dev/mt/c*x*t*y*d*z*options*** (logical tape device) **/dev/rmt/*x*m*options*** (or) **/dev/rmt/c*x*t*y*d*z*options*** (raw tape device) (Choice of format is outside the scope of this document. The *options* include **BEST**, **h** for high density, **m** for medium density, **l** for low density, **c** for compression, **n** for norewind, and/or **b** for Berkley style access. Examples are: **/dev/mt/*x*m** **/dev/rmt/*x*mn** **/dev/mt/*x*mb** **/dev/rmt/*x*mnb** **/dev/mt/c*x*t*y*d*z*BESTn** **/dev/rmt/ c*x*t*y*d*z*BESTb** **/dev/mt/ c*x*t*y*d*z*BESTnb)**	**/dev/ac/c*x*t*y*d*z* _*xa*** (block device where *xa* is one of **1a**, **1b**, **2a**, ..., **31a**, **32b**) **/dev/rac/c*x*t*y*d*z* _*xa*** (character device where *xa* is one of **1a**, **1b**, **2a**, ..., **31a**, **32b**)

[a] In logical unit name format samples, text appearing in **bold weight** shall appear in the shown position as it appears in the format sample. Text appearing in *normal weight italics* is a placeholder for similar text determined by the rules of the OS. Italicized lower case *x* or *y* or *z* represents any decimal number and may be more than one digit. Italicized lower case *a* represents one or two lower case alphabetic characters. Italicized lower case *b* represents the null terminated string created from the designator/identifier field associated with the logical unit in the Device Identification VPD page (see SPC-4). Normal text in parentheses is descriptive, not format sample.

[b] These names shall reference the raw (i.e., unformatted) and unpartitioned disk. So long as it is consistent with relevant Solaris OS documentation, rdsk shall be used to indicate the raw device and s2 shall be used to reference the unpartitioned disk. Should other formats be established by relevant OS documentation for representing these characteristics, the new formats shall also be considered preferred formats.

Table 4 — Preferred format for logical unit OSDeviceName (part 3 of 4)

OS	Preferred format for logical unit type [a]			
	disk/optical	**cd-rom**	**tape**	**changer**
HP-UX For version 11i V3	**/dev/disk/disk***x* (Block device where *x* is the instance number of the device.) **/dev/rdisk/disk***x* (Character device where *x* is the instance number of the device.)	**/dev/disk/disk***x* (Block device where *x* is the instance number of the device.) **/dev/rdisk/disk***x* (Character device where *x* is the instance number of the device.)	**/dev/rtape/tape***x***_BEST***a* (Where *x* is the instance number of the device and *a* represents the options of **n** for norewind and/or **b** for Berkley-style access.) Examples are: **/dev/rtape/tape***x***_BEST** **/dev/rtape/tape***x***_BESTn** **/dev/rtape/tape***x***_BESTb** **/dev/rtape/tape***x***_BESTnb**	**/dev/rchgr/chgr***x* (where *x* is the instance number of the device.)
OpenVMS	**$***y***$DGA***x***:** (FC or external SAS device where *y* is the device allocation class and *x* is the device number e.g. **1DGA010:** is device 10 of allocation class 1)	**$***y***$DGA***x***:** (FC device where *y* is the device allocation class and *x* is the device number e.g. **1DGA010:** is device 10 of allocation class 1)	**$***y***$MGA***x***:** (FC or external SAS device where *y* is the device allocation class and *x* is the device number e.g. **1MGA010:** is device 10 of allocation class 1)	**$***y***$GGA***x***:** (FC device where *y* is the device allocation class and *x* is the device number e.g. **1GGA010:** is device 10 of allocation class 1)

[a] In logical unit name format samples, text appearing in **bold weight** shall appear in the shown position as it appears in the format sample. Text appearing in *normal weight italics* is a placeholder for similar text determined by the rules of the OS. Italicized lower case *x* or *y* or *z* represents any decimal number and may be more than one digit. Italicized lower case *a* represents one or two lower case alphabetic characters. Italicized lower case *b* represents the null terminated string created from the designator/identifier field associated with the logical unit in the Device Identification VPD page (see SPC-4). Normal text in parentheses is descriptive, not format sample.

[b] These names shall reference the raw (i.e., unformatted) and unpartitioned disk. So long as it is consistent with relevant Solaris OS documentation, rdsk shall be used to indicate the raw device and s2 shall be used to reference the unpartitioned disk. Should other formats be established by relevant OS documentation for representing these characteristics, the new formats shall also be considered preferred formats.

Table 4 — Preferred format for logical unit OSDeviceName (part 4 of 4)

OS	Preferred format for logical unit type [a]			
	disk/optical	**cd-rom**	**tape**	**changer**
Tru64 Version 5.1	**/dev/disk/dsk***xa* (block device where *x* represents the logical device number and *a* represents the partition, one of **a** thru **h**) **/dev/rdisk/dsk***xa* (raw device where *x* represents the logical device number and *a* represents the partition, one of **a** thru **h**)	**/dev/disk/dsk***xa* (block device where *x* represents the logical device number and *a* represents the partition, one of **a** thru **h**) **/dev/rdisk/dsk***xa* (raw device where *x* represents the logical device number and *a* represents the partition, one of **a** thru **h**)	**/dev/tape/tape***x***d***ya* (Rewind tape device before use where *x* represents the logical device number, *y* represents the density setting, one of **0** through **7**, and *a* optional compression indicated by presense of the letter **c**) **/dev/tape/ntape***x***d***ya* (Don't rewind tape device before use where *x* represents the logical device number, *y* represents the density setting, one of **0** through **7**, and *a* optional compression indicated by presense of the letter **c**)	**/dev/changer/mc***x* (media changer device where *x* represents the logical device number)
vSphere	**/dev/disks/***b*	(zero length string)	(zero length string)	(zero length string)

[a] In logical unit name format samples, text appearing in **bold weight** shall appear in the shown position as it appears in the format sample. Text appearing in *normal weight italics* is a placeholder for similar text determined by the rules of the OS. Italicized lower case *x* or *y* or *z* represents any decimal number and may be more than one digit. Italicized lower case *a* represents one or two lower case alphabetic characters. Italicized lower case *b* represents the null terminated string created from the designator/identifier field associated with the logical unit in the Device Identification VPD page (see SPC-4). Normal text in parentheses is descriptive, not format sample.

[b] These names shall reference the raw (i.e., unformatted) and unpartitioned disk. So long as it is consistent with relevant Solaris OS documentation, rdsk shall be used to indicate the raw device and s2 shall be used to reference the unpartitioned disk. Should other formats be established by relevant OS documentation for representing these characteristics, the new formats shall also be considered preferred formats.

6.10.2.10 ScsiBusNumber

Within the context of any FCP-4 target port or SSP target port attribute data structures defined in 6.10.1, the ScsiBusNumber field shall contain zero or a number that in accord with the specifications of the operating system identifies the SCSI Domain in which the operating system represents a SCSI logical unit to application programs. This may be referenced as bus number in OS documentation (see SAM-5 and relevant OS documentation).

Within the context of a target mapping returned from an SM-HBA-2 API function, if the driver for the HBA that returns the target mapping has registered with the operating system for a local bus number, the ScsiBusNumber field shall contain the registered local bus number.

6.10.2.11 ScsiTargetNumber

Within the context of any FCP-4 target port or SSP target port attribute data structures defined in 6.10.1, the ScsiTargetNumber field shall contain zero or a number that in accord with the specifications of the operating system identifies the SCSI target device in which the operating system may represent SCSI logical units to application programs. This may be referenced as target ID or device number in OS documentation (see SAM-5 and relevant OS documentation).

Within the context of a target mapping returned from an SM-HBA-2 API function, the ScsiTargetNumber field shall contain the OS target ID of the device that is mapped.

6.10.2.12 ScsiOSLun

Within the context of any FCP-4 target port or SSP target port attribute data structures defined in 6.10.1, the ScsiOSLun field shall contain zero or a number that in accord with the specifications of the operating system distinguishes a SCSI logical unit within its represented device to application programs (see SAM-5 and relevant OS documentation).

Within the context of a target mapping returned from an SM-HBA-2 API function, the ScsiOSLun field shall contain the OS LUN of the logical unit that is mapped.

6.10.3 Persistent Binding Capabilities

6.10.3.1 Persistent Binding Capability: SMHBA_CAN_BIND_TO_WWPN

The persistent binding capability SMHBA_CAN_BIND_TO_WWPN shall indicate the ability of an HBA to accept a persistent binding that identifies the 64 bit SCSI LUN of a SCSI Logical Unit (see SAM-5) and one of the following:

a) FCP-4 target port by its Name_Identifier
b) SSP target port by its Port_Identifier.

6.10.3.2 Persistent Binding Capability: SMHBA_CAN_BIND_TO_LUID

The persistent binding capability SMHBA_CAN_BIND_TO_LUID shall indicate the ability of an HBA to accept a persistent binding that identifies one of the following:

a) FCP-4 target logical unit by the value of one of its LUIDs ; or
b) SSP target logical unit by the value of one of its LUIDs.

6.10.3.3 Persistent Binding Capability: SMHBA_CAN_BIND_ANY_LUNS

The persistent binding capability SMHBA_CAN_BIND_ANY_LUNS shall indicate the ability of an HBA to accept persistent binding settings that independently specify both the ScsiOSLuns and TargetLuns.

An HBA that does not express the SMHBA_CAN_BIND_ANY_LUNS capability may require that for any pair of persistent binding settings, the HBA may be able to support them both concurrently only if:

a) the OS target number identified by both persistent bindings is the same and the FCP-4 target port or the SSP target port identified by both persistent bindings is the same; or
b) the OS target number identified by the persistent bindings is different and the FCP-4 target port or the SSP target port identified by the persistent bindings is different.

(I.e., For an HBA that does not express the SMHBA_CAN_BIND_ANY_LUNS, all persistent binding settings shall preserve the groupings of logical units into devices.)

NOTE 19 - In many OS implementations unpredictable behavior, possibly including failure to boot, may result from mapping OS LUN 0 to any FCP LUN other than 0.

6.10.3.4 Persistent Binding Capability: SMHBA_CAN_ BIND_AUTOMAP

The persistent binding capability SMHBA_CAN_BIND_AUTOMAP shall indicate the ability of an HBA to attempt to automatically generate target mappings and persistent bindings for all discovered storage resources.

If this capability is not indicated or disabled, target mappings shall be established only based on persistent bindings that have been explicitly configured either by the SMHBA_SetPersistentBinding function (see TBD) or by a vendor specific method.

NOTE 20 - Disabling this capability is sometimes described as HBA-based LUN Masking.

6.10.4 Persistent Binding Setting Types

6.10.4.1 Persistent Binding Type: SMHBA_BIND_TO_WWPN

If a persistent binding setting includes this feature in its type, the setting shall identify the 64 bit SCSI LUN of a SCSI Logical Unit (see SAM-5) and one of the following:

a) FCP-4 target port by its PortWWN field; or,
b) SSP target port by its Port_Identifier field

The LUID fields shall be ignored.

If a persistent binding setting includes more than one of SMHBA_BIND_TO_WWPN and SMHBA_BIND_TO_LUID in its type, that setting shall be rejected.

6.10.4.2 Persistent Binding Type: SMHBA_BIND_TO_LUID

If a persistent binding setting includes this feature in its type, the setting shall identify the FCP-4 target logical unit or SSP target logical unit by its LUID field. The PortWWN field shall be ignored.

If a persistent binding setting includes more than one of SMHBA_BIND_TO_WWPN and SMHBA_BIND_TO_LUID in its type, that setting shall be rejected.

6.11 Asynchronous Event Notification Attributes

6.11.1 Asynchronous Event Data Declarations

6.11.1.1 Callback Handle

```
typedef void *HBA_CALLBACKHANDLE;
```

6.11.1.2 HBA Add Category Event Type

```
#define       HBA_EVENT_ADAPTER_ADD             0x101
```

6.11.1.3 HBA Category Event Types

```
#define       HBA_EVENT_ADAPTER_UNKNOWN          0x100
#define       HBA_EVENT_ADAPTER_REMOVE           0x102
#define       HBA_EVENT_ADAPTER_CHANGE           0x103
```

6.11.1.4 Port Category Event Types

```
#define       HBA_EVENT_PORT_UNKNOWN             0x200
#define       HBA_EVENT_PORT_OFFLINE             0x201
#define       HBA_EVENT_PORT_ONLINE              0x202
#define       HBA_EVENT_PORT_NEW_TARGETS         0x203
#define       HBA_EVENT_PORT_FABRIC              0x204
#define       HBA_EVENT_PORT_BROADCAST_CHANGE    0x205
#define       HBA_EVENT_PORT_BROADCAST_SES       0x208
#define       HBA_EVENT_PORT_BROADCAST_RSVCHG0   0x206
#define       HBA_EVENT_PORT_BROADCAST_RSVCHG1   0x207
#define       HBA_EVENT_PORT_BROADCAST_EXPANDER  0x209
#define       HBA_EVENT_PORT_BROADCAST_AEVENT    0x20A
#define       HBA_EVENT_PORT_BROADCAST_RSV3      0x20B
#define       HBA_EVENT_PORT_BROADCAST_RSV4      0x20C
#define       HBA_EVENT_PORT_ALL                 0x2FF
```

6.11.1.5 Port Statistics Category Event Types

```
#define       HBA_EVENT_PORT_STAT_THRESHOLD      0x301
#define       HBA_EVENT_PORT_STAT_GROWTH         0x302
```

6.11.1.6 Phy Statistics Category Event Types

```
#define       HBA_EVENT_PHY_STAT_THRESHOLD       0x351
#define       HBA_EVENT_PHY_STAT_GROWTH          0x352
```

6.11.1.7 Target Category Event Types

```
#define       HBA_EVENT_TARGET_UNKNOWN           0x400
#define       HBA_EVENT_TARGET_OFFLINE           0x401
#define       HBA_EVENT_TARGET_ONLINE            0x402
#define       HBA_EVENT_TARGET_REMOVED           0x403
```

6.11.1.8 Link Category Event Types

```
#define       HBA_EVENT_LINK_UNKNOWN             0x500
#define       HBA_EVENT_LINK_INCIDENT            0x501
```

6.11.2 Asynchronous Event Attribute Specifications

6.11.2.1 EventType

EventType shall be set to a value from table 5 indicating an event reported by the asynchronous event API (see 7.9.1.1).

Table 5 — Asynchronous event type codes (part 1 of 2)

EventType value	Indicated event
HBA_EVENT_ADAPTER_ADD	An HBA supported by the HBA API has been added to the local system.
HBA_EVENT_ADAPTER_REMOVE	An HBA supported by the HBA API has been removed from the local system.
HBA_EVENT_ADAPTER_CHANGE	There has been a configuration change to an HBA on the local system supported by the HBA API.
HBA_EVENT_PORT_OFFLINE	An HBA on the local system supported by the HBA API has stopped providing communication.
HBA_EVENT_PORT_ONLINE	An HBA on the local system supported by the HBA API has restarted providing communication.
HBA_EVENT_PORT_NEW_TARGETS	An HBA on the local system supported by the HBA API has added a FCP target port, or a SSP target port.
HBA_EVENT_PORT_FABRIC	An HBA on the local system supported by the HBA API has received an RSCN ELS.
HBA_EVENT_PORT_BROADCAST_CHANGE	An HBA on the local system supported by the SM-HBA-2 API has received a SAS Broadcast (CHANGE) primitive (see SPL).
HBA_EVENT_PORT_BROADCAST_SES	An HBA on the local system supported by the SM-HBA-2 API has received a SAS BROADCAST (SES) primitive (see SPL).
HBA_EVENT_PORT_BROADCAST_RSVCHG0	An HBA on the local system supported by the SM-HBA-2 API has received a SAS BROADCAST (RESERVED CHANGE 0) primitive (see SPL).
HBA_EVENT_PORT_BROADCAST_RSVCHG1	An HBA on the local system supported by the SM-HBA-2 API has received a SAS BROADCAST (RESERVED CHANGE 1) primitive (see SPL).
HBA_EVENT_PORT_BROADCAST_EXPANDER	An HBA on the local system supported by the SM-HBA-2 API has received a SAS BROADCAST (EXPANDER) primitive (see SPL).
HBA_EVENT_PORT_BROADCAST_AEVENT	An HBA on the local system supported by the SM-HBA-2 API has received a SAS BROADCAST (ASYNCHRONOUS EVENT) primitive (see SPL).

Table 5 — Asynchronous event type codes (part 2 of 2)

EventType value	Indicated event
HBA_EVENT_PORT_BROADCAST_RSV3	An HBA on the local system supported by the SM-HBA-2 API has received a SAS BROADCAST (RESERVED 3) primitive (see SPL).
HBA_EVENT_PORT_BROADCAST_RSV4	An HBA on the local system supported by the SM-HBA-2 API has received a SAS BROADCAST (RESERVED 4) primitive (see SPL).
HBA_EVENT_PORT_STAT_THRESHOLD	A statistical counter for a specified port on the HBA has reached a registered level.
HBA_EVENT_PORT_STAT_GROWTH	A statistical counter for a specified port has increased at a rate equal to or in excess of a registered rate.
HBA_EVENT_PHY_STAT_THRESHOLD	A statistical counter for a specified phy has reached a registered level.
HBA_EVENT_PHY_STAT_GROWTH	A statistical counter for a specified phy has increased at a rate equal to or in excess of a registered rate.
HBA_EVENT_TARGET_OFFLINE	Operational use of an FCP-4 target device or SSP target device supported by the HBA API has become impossible.
HBA_EVENT_TARGET_ONLINE	Operational use of an FCP-4 target device or SSP target device supported by the HBA API has been restored.
HBA_EVENT_TARGET_REMOVED	An FCP-4 target device or SSP target device supported by the HBA API has been removed from the fabric.
HBA_EVENT_LINK_INCIDENT	An HBA on the local system supported by the HBA API has received an RLIR ELS.

6.12 Library Attributes

6.12.1 Library Attribute Data Declarations

Functions implemented in compliance with this standard shall conform to the function prototypes declared in subclause 6.12.

6.12.2 Function Prototypes

The following are prototypes for the functions spcified in SM-HBA-2 API libraries.

```
typedef HBA_UINT32   (* SMHBA2GetVersionFunc)();
typedef HBA_STATUS   (* HBALoadLibraryFunc)();
typedef HBA_STATUS   (* HBAFreeLibraryFunc)();
typedef HBA_UINT32   (* HBAGetNumberOfAdaptersFunc)();
typedef HBA_UINT32   (* SMHBA2RegisterLibraryFunc)(SMHBA2_ENTRYPOINTS *);
typedef HBA_UINT32   (* SMHBAGetWrapperLibraryAttributesFunc)
                     (SMHBA_LIBRARYATTRIBUTES *);
typedef HBA_UINT32   (* SMHBAGetVendorLibraryAttributesFunc)
                     (HBA_UINT32, SMHBA_LIBRARYATTRIBUTES *);

typedef HBA_STATUS   (* SMHBA2GetAdapterHandleByIndexFunc)
                     (HBA_UINT32, HBA_HANDLE *);
typedef HBA_STATUS   (* SMHBA2GetAdapterAttributesFunc)
                     (HBA_HANDLE, SMHBA2_ADAPTERATTRIBUTES *);
typedef HBA_STATUS   (* SMHBAGetNumberOfPortsFunc)
                     (HBA_HANDLE, HBA_UINT32 *);
typedef HBA_STATUS   (* SMHBA2GetAdapterBusAddressFunc)
                     (HBA_HANDLE,SMHBA2_BUSADDRESS *);
typedef HBA_STATUS   (* SMHBA2GetPortTypeFunc)
                     (HBA_HANDLE, SMHBA2_PORTTYPE *);
typedef HBA_STATUS   (* SMHBA2GetPortAttributesFunc)
                     (HBA_HANDLE, SMHBA2_PORTATTRIBUTES *);
typedef HBA_STATUS   (* SMHBA2GetPortAttributesByWWNFunc)
                     (HBA_HANDLE, HBA_WWN, HBA_WWN, SMHBA2_PORTATTRIBUTES *);
typedef HBA_STATUS   (* SMHBA2GetPhyTypeFunc)
                     (HBA_HANDLE, SMHBA2_PHYTYPE *);
typedef HBA_STATUS   (* SMHBA2GetPhyAttributesFunc)
                     (HBA_HANDLE, SMHBA2_PHYAATRIBUTES *);
typedef HBA_STATUS   (* SMHBA2GetPhyCtlrAttributesFunc)
                     (HBA_HANDLE, SMHBA2_NPCATTRIBUTES *);
typedef HBA_STATUS   (* SMHBA2GetFabricInfoFunc)
                     (HBA_HANDLE, SMHBA2_FABRICINFO *);

typedef HBA_STATUS   (* SMHBA2GetPortsOnAdapterFunc)
                     (HBA_HANDLE, SMHBA_HANDLELIST *);
typedef HBA_STATUS   (* SMHBA2GetAdapterForPortFunc)
                     (HBA_HANDLE, HBA_HANDLE *);
typedef HBA_STATUS   (* SMHBA2GetLEPForPortFunc)
                     (HBA_HANDLE, SMHBA2_FCOE_LEP *);
typedef HBA_STATUS   (* SMHBA2GetDiscoveredPortsFunc)
                     (HBA_HANDLE, SMHBA_HANDLELIST *);
typedef HBA_STATUS   (* SMHBA2GetPhysOnAdapterFunc)
                     (HBA_HANDLE, SMHBA_HANDLELIST *);
typedef HBA_STATUS   (* SMHBA2GetAdapterForPhyFunc)
                     (HBA_HANDLE, HBA_HANDLE *);
typedef HBA_STATUS   (* SMHBA2GetPortsOnPhyFunc)
                     (HBA_HANDLE, SMHBA_HANDLELIST *);
typedef HBA_STATUS   (* SMHBA2GetPhysForPortFunc)
                     (HBA_HANDLE, SMHBA_HANDLELIST *);
typedef HBA_STATUS   (* SMHBA2GetCtlrForPhyFunc)
                     (HBA_HANDLE, HBA_HANDLE *);
typedef HBA_STATUS   (* SMHBA2GetPhyForCtlrFunc)
                     (HBA_HANDLE, HBA_HANDLE *);
typedef HBA_STATUS   (* SMHBA2GetFabricsForCtlrFunc)
                     (HBA_HANDLE, HBA_HANDLELIST *);
```

```
typedef HBA_STATUS   (* SMHBA2GetCtlrsForFabricFunc)
                     (HBA_HANDLE, HBA_HANDLELIST *);
typedef HBA_STATUS   (* SMHBA2GetFabricForPortFunc)
                     (HBA_HANDLE, HBA_HANDLE *);
typedef HBA_STATUS   (* SMHBA2GetPortsForFabricFunc)
                     (HBA_HANDLE, HBA_HANDLELIST *);

typedef HBA_STATUS   (* SMHBA2GetPortStatisticsFunc)
                     (HBA_HANDLE, SMHBA_PORTSTATISTICS *);
typedef HBA_STATUS   (* SMHBA2GetProtocolStatisticsFunc)
                     (HBA_HANDLE, HBA_UINT32, SMHBA_PROTOCOLSTATISTICS *);
typedef HBA_STATUS   (* SMHBA2GetPhyStatisticsFunc)
                     (HBA_HANDLE, SMHBA2_PHYSTATISTICS *);
typedef HBA_STATUS   (* SMHBA2GetFIPStatisticsFunc)
                     (HBA_HANDLE, SMHBA2_FIPSTATISTICS *);

typedef HBA_STATUS   (* HBASendCTPassThruV2Func)
                     (HBA_HANDLE, HBA_WWN, void *, HBA_UINT32, void *,
                     HBA_UINT32 *);
typedef HBA_STATUS   (* HBASetRNIDMgmtInfoFunc)
                     (HBA_HANDLE, HBA_MGMTINFO *);
typedef HBA_STATUS   (* HBAGetRNIDMgmtInfoFunc)
                     (HBA_HANDLE, HBA_MGMTINFO *);
typedef HBA_STATUS   (* HBASendRNIDV2Func)
                     (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_UINT32, HBA_UINT32, void *,
                     HBA_UINT32*);
typedef HBA_STATUS   (* HBASendSRLFunc)
                     (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_UINT32, void *,
                     HBA_UINT32 *);
typedef HBA_STATUS   (* HBASendLIRRFunc)
                     (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_UINT8, HBA_UINT8,
                     void *, HBA_UINT32 *);
typedef HBA_STATUS   (* HBASendRLSFunc)
                     (HBA_HANDLE, HBA_WWN, HBA_WWN, void *, HBA_UINT32 *);
typedef HBA_STATUS   (* SMHBASendTESTFunc)
                     (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_UINT32, void *, HBA_UINT32);
typedef HBA_STATUS   (* SMHBASendECHOFunc)
                     (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_UINT32, void *, HBA_UINT32,
                     void *, HBA_UINT32 *);
typedef HBA_STATUS   (* SMHBASendSMPPassThruFunc)
                     (HBA_HANDLE, HBA_WWN, HBA_WWN, void *, HBA_UINT32, void *,
                     HBA_UINT32 *);

typedef HBA_STATUS   (* SMHBAGetBindingCapabilityFunc)
                     (HBA_HANDLE, HBA_WWN, SMHBA_BIND_CAPABILITY *);
typedef HBA_STATUS   (* SMHBAGetBindingSupportFunc)
                     (HBA_HANDLE, HBA_WWN, SMHBA_BIND_CAPABILITY *);
typedef HBA_STATUS   (* SMHBASetBindingSupportFunc)
                     (HBA_HANDLE, HBA_WWN, SMHBA_BIND_CAPABILITY);
typedef HBA_STATUS   (* SMHBAGetTargetMappingFunc)
                     (HBA_HANDLE, HBA_WWN, SMHBA_TARGETMAPPING *);
typedef HBA_STATUS   (* SMHBAGetPersistentBindingFunc)
                     (HBA_HANDLE, HBA_WWN, SMHBA_BINDING *);
typedef HBA_STATUS   (* SMHBASetPersistentBindingFunc)
                     (HBA_HANDLE, HBA_WWN, const SMHBA_BINDING *);
typedef HBA_STATUS   (* SMHBARemovePersistentBindingFunc)
                     (HBA_HANDLE, HBA_WWN, const SMHBA_BINDING *);
typedef HBA_STATUS   (* SMHBARemoveAllPersistentBindingsFunc)
```

```
                    (HBA_HANDLE, HBA_WWN);
typedef HBA_STATUS  (* SMHBAGetLUNStatisticsFunc)
                    (HBA_HANDLE, const HBA_SCSIID *, SMHBA_PROTOCOLSTATISTICS *);

typedef HBA_STATUS  (* SMHBARegisterForAdapterAddEventsFunc)
                    (void (*)(void *, HBA_WWN, HBA_UINT32),
                    void *, HBA_CALLBACKHANDLE *);
typedef HBA_STATUS  (* SMHBARegisterForAdapterEventsFunc)
                    (void (*)(void *, HBA_WWN, HBA_UINT32),
                    void *, HBA_HANDLE, HBA_CALLBACKHANDLE *);
typedef HBA_STATUS  (* SMHBARegisterForAdapterPortEventsFunc)
                    (void (*)(void *, HBA_WWN, HBA_UINT32, HBA_UINT32),
                     void *, HBA_HANDLE, HBA_WWN, HBA_UINT32, HBA_CALLBACKHANDLE *);
typedef HBA_STATUS  (* SMHBA2RegisterForAdapterPhyStatEventsFunc)
                    (void (*)(void *, HBA_HANDLE, HBA_UINT32),
                    void *, HBA_HANDLE, SMHBA2_PHYSTATISTICS,
                    HBA_UINT32, HBA_CALLBACKHANDLE *);
typedef HBA_STATUS  (* SMHBARegisterForTargetEventsFunc)
                    (void (*)(void *, HBA_WWN, HBA_WWN, HBA_WWN, HBA_UINT32),
                     void *, HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_WWN,
                     HBA_CALLBACKHANDLE *, HBA_UINT32);
typedef HBA_STATUS  (* HBARegisterForLinkEventsFunc)
                    (void (*)(void *, HBA_WWN, HBA_UINT32, void *, HBA_UINT32),
                    void *, void *, HBA_UINT32, HBA_HANDLE, HBA_CALLBACKHANDLE *);
typedef HBA_STATUS  (* HBARemoveCallbackFunc)
                    (HBA_CALLBACKHANDLE);

typedef HBA_STATUS  (* SMHBAScsiInquiryFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_WWN, SMHBA_SCSILUN, HBA_UINT8,
                     HBA_UINT8, void *, HBA_UINT32 *, HBA_UINT8 *,
                     void *, HBA_UINT32 *);
typedef HBA_STATUS  (* SMHBAScsiReportLunsFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_WWN, void *, HBA_UINT32 *,
                     HBA_UINT8 *, void *, HBA_UINT32 *);
typedef HBA_STATUS  (* SMHBAScsiReadCapacityFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_WWN, SMHBA_SCSILUN, void *,
                    HBA_UINT32 *, HBA_UINT8 *, void *, HBA_UINT32 *);
typedef HBA_STATUS  (* SMHBA_ScsiManagementInFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_WWN, SMHBA_SCSILUN,
                    HBA_UINT8, HBA_UINT16, HBA_UINT8,
                    * void, * HBA_UINT32, * HBA_UINT8, * void, * HBA_UINT32);
typedef HBA_STATUS  (* SMHBA_ScsiManagementOutFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_WWN, SMHBA_SCSILUN,
                    HBA_UINT8, HBA_UINT16, HBA_UINT8,
                    * void, HBA_UINT32, * HBA_UINT8, * void, * HBA_UINT32);
```

6.12.3 SM-HBA-2 Entry Point Data Declarations

The following structure is used to register a vendor-specific library that is compliant with annex A with the wrapper library.

```
typedef struct SMHBA2_EntryPoints {
       SMHBA2GetVersionFunc                    GetVersionHandler;
       HBALoadLibraryFunc                      LoadLibraryHandler;
       HBAFreeLibraryFunc                      FreeLibraryHandler;
       HBAGetNumberOfAdaptersFunc              GetNumberOfAdaptersHandler;
       SMHBAGetVendorLibraryAttributesFunc     GetVendorLibraryAttributesHandler;

       SMHBA2GetAdapterHandleByIndexFunc       GetAdapterHandleByIndexHandler;
```

```
SMHBA2GetAdapterAttributesFunc          GetAdapterAttributesHandler;
SMHBAGetNumberOfPortsFunc               GetNumberOfPortsHandler;
SMHBA2GetAdapterBusAddressFunc          GetAdapterBusAttributesHandler;
SMHBA2GetPortTypeFunc                   GetPortTypeHandler;
SMHBA2GetPortAttributesFunc             GetAdapterPortAttributesHandler;
SMHBA2GetPortAttributesByWWNFunc        GetPortAttributesByWWNHandler;
SMHBA2GetPhyTypeFunc                    GetPhyTypeHandler;
SMHBA2GetPhyAttributesFunc              GetPhyAttributesHandler;
SMHBA2GetPhyCtlrAttributesFunc          GetCtlrAttributesHandler;
SMHBA2GetFabricInfoFunc                 GetFabricInfoHandler;

SMHBA2GetPortsOnAdapterFunc             GetPortsOnAdapterHandler;
SMHBA2GetAdapterForPortsFunc            GetAdapterForPorts;
SMHBA2GetLEPForPortFunc                 GetLEPForPortHandler;
SMHBA2GetDiscoveredPortsFunc            GetDiscoveredPortsHandler;
SMHBA2GetPhysOnAdapterFunc              GetPhysOnAdapterHandler;
SMHBA2GetAdapterForPhyFunc              GetAdapterForPhyHandler;
SMHBA2GetPortsOnPhyFunc                 GetPortsOnPhyHandler;
SMHBA2GetPhysForPortFunc                GetPhyForPortHandler;
SMHBA2GetCltrForPhyFunc                 GetCltrForPhyHandler;
SMHBA2GetPhyForCtlrFunc                 GetPhyForCltrHandler;
SMHBA2GetFabricsForCltrFunc             GetFabricsForCltrHandler;
SMHBA2GetCltrsForFabricFunc             GetCltrsForFabricHandler;
SMHBA2GetFabricForPortFunc              GetFabricForPortHandler;
SMHBA2GetPortsForFabricFunc             GetPortsForFabric;

SMHBA2GetPortStatisticsFunc             GetPortStatisticsHandler;
SMHBA2GetProtocolStatisticsFunc         GetProtocolStatisticsHandler;
SMHBAG2etPhyStatisticsFunc              GetPhyStatisticsHandler;
SMHBAG2etFIPStatisticsFunc              GetFIPStatisticsHandler;

HBASendCTPassThruV2Func                 SendCTPassThruV2Handler;
HBASetRNIDMgmtInfoFunc                  SetRNIDMgmtInfoHandler;
HBAGetRNIDMgmtInfoFunc                  GetRNIDMgmtInfoHandler;
HBASendRNIDV2Func                       SendRNIDV2Handler;
HBASendSRLFunc                          SendSRLHandler;
HBASendLIRRFunc                         SendLIRRHandler;
HBASendRLSFunc                          SendRLSHandler;
SMHBASendTESTFunc                       SendTESTHandler;
SMHBASendECHOFunc                       SendEchoHandler;
SMHBASendSMPPassThruFunc                SendSMPPassThruHandler;

SMHBAGetBindingCapabilityFunc           GetBindingCapabilityHandler;
SMHBAGetBindingSupportFunc              GetBindingSupportHandler;
SMHBASetBindingSupportFunc              SetBindingSupportHandler;
SMHBAGetTargetMappingFunc               GetTargetMappingHandler;
SMHBAGetPersistentBindingFunc           GetPersistentBindingHandler;
SMHBASetPersistentBindingFunc           SetPersistentBindingHandler;
SMHBARemovePersistentBindingFunc        RemovePersistentBindingHandler;
SMHBARemoveAllPersistentBindingsFunc    RemoveAllPersistentBindingsHandler;
SMHBAGetLUNStatisticsFunc               GetLUNStatisticsHandler;

SMHBAScsiInquiryFunc                    ScsiInquiryHandler;
SMHBAScsiReportLunsFunc                 ScsiReportLUNsHandler;
SMHBAScsiReadCapacityFunc               ScsiReadCapacityHandler;
SMHBA_ScsiManagementInFunc              ScsiManagementInHandler;
SMHBA_ScsiManagementOutFunc             ScsiManagementOutHandler;

SMHBARegisterForAdapterAddEventsFunc    RegisterForAdapterAddEventsHandler;
```

```
       SMHBARegisterForAdapterEventsFunc          RegisterForAdapterEventsHandler;
       SMHBARegisterForAdapterPortEventsFunc      RegisterForAdapterPortEventsHandler;
       SMHBARegisterForAdapterPortStatEventsFunc
                                                  RegisterForAdapterPortStatEventsHandler;
       SMHBA2RegisterForAdapterPhyStatEventsFunc
                                                  RegisterForAdapterPhyStatEventsHandler;
       SMHBARegisterForTargetEventsFunc           RegisterForTargetEventsHandler;
       HBARegisterForLinkEventsFunc               RegisterForLinkEventsHandler;
       HBARemoveCallbackFunc                      RemoveCallbackHandler;
} SMHBA2_ENTRYPOINTS, *PSMHBA2_ENTRYPOINTS;
```

6.12.4 Entry Point Specifications

This structure shall be filled with the entry addresses of the vendor specific implementation of all library functions and returned to the wrapper library (see 7.2.4).

6.12.5 Library Attribute Data Declarations

```
typedef struct SMHBA_LibraryAttributes {
       char          LibPath[256];
       char          VName[256];
       char          VVersion[256];
       struct {
              int           tm_mday;      /* day of the month - [1 - 31] */
              int           tm_mon;       /* months since January - [0 - 11] */
              int           tm_year;      /* years since 1900 */
       } build_date;
} SMHBA_LIBRARYATTRIBUTES, *PSMHBA_LIBRARYATTRIBUTES;
```

6.12.6 Library Attribute Specifications

6.12.6.1 Compliance

Requirements are specified in annex A for support of the attributes specified in 6.12.6.

6.12.6.2 LibPath

LibPath shall be an ASCII string the value of which is the fully qualified path name of the library file.

6.12.6.3 VName

VName shall be an ASCII string the value of which is the name of the organization that developed the library code.

6.12.6.4 VVersion

VVersion shall be an ASCII string the value of which is the Identification used by the developing organization for the code revision of the library being called represented as a null-terminated ASCII string.

6.12.6.5 build_date

The content of build_date shall be the structure containing the date at which the developing organization completed the code revision of the library being called. The time zone reference for the build_date structure shall be the Coordinated Universal Time (UCT). Zero values are acceptable for fields beyond the intended resolution of the developer.

6.13 FC-3 Management Attributes

6.13.1 FC-3 Management Data Declarations

```
typedef enum HBA_wwntype {NODE_WWN, PORT_WWN} HBA_WWNTYPE;

typedef struct HBA_MgmtInfo {
       HBA_WWN              wwn;
       HBA_UINT32           unittype;
       HBA_UINT32           PortId;
       HBA_UINT32           NumberOfAttachedNodes;
       HBA_UINT16           IPVersion;
       HBA_UINT16           UDPPort;
       HBA_UINT8            IPAddress[16];
       HBA_UINT16           reserved;
       HBA_UINT16           TopologyDiscoveryFlags;
} HBA_MGMTINFO, *PHBA_MGMTINFO;
```

6.13.2 FC-3 Management Attribute Overview

Although the HBA_MgmtInfo structure closely resembles the Specific Identification Data in an RNID Accept with Node Identification Data Format DFh (see FC-LS-2), it is different (i.e., it includes only 8 bytes of the initial 16 bytes of the Specific Identification Data, the names of the fields in this structure reflect an earlier version of the reply to the RNID ELS, and RNID was significantly redefined in FC-LS-2 after the predecessor to this standard had stabilized).

6.13.3 FC-3 Management Attribute Specifications

6.13.3.1 FC-3 Management Requirements

Requirements are specified in annex A for support of the attributes specified in 6.13.3.

6.13.3.2 WWN

The WWN field of a data structure of type HBA_MGMTINFO shall contain the value of the first eight bytes of the initial 16 bytes of the Specific Identification Data in an RNID Accept with the Node Identification Data Format set to DFh (see FC-LS-2). The value of the WWN field is vendor specific data and may not be a Worldwide_Name.

6.13.3.3 unittype

The unittype field of a data structure of type HBA_MGMTINFO shall contain the value of the Association Type field of the Specific Identification Data in an RNID Accept with the Node Identification Data Format set to DFh (see FC-LS-2), describing the type of equipment this HBA represents.

> NOTE 21 - The RNID Association Type field was identified as the Unit Type field in early drafts of its specification that were concurrent with the original specification of this field.

6.13.3.4 PortId

The PortId field of a data structure of type HBA_MGMTINFO shall contain the value of the Physical Port Number field of the Specific Identification Data in an RNID Accept with the Node Identification Data Format set to DFh (see FC-LS-2).

6.13.3.5 NumberOfAttachedNodes

The NumberOfAttachedNodes field of a data structure of type HBA_MGMTINFO shall contain the value of the Number of Attached Nodes field of the Specific Identification Data in an RNID Accept with the Node Identification Data Format set to DFh (see FC-LS-2).

6.13.3.6 IPVersion

The IPVersion field of a data structure of type HBA_MGMTINFO shall contain the value of the concatenated Node Management and IP Version fields of the Specific Identification Data in an RNID Accept with the Node Identification Data Format set to DFh (see FC-LS-2), indicating the management protocol stack and whether the following IP address is an IPv4 address (see RFC 791) or an IPv6 address (see RFC 2460).

6.13.3.7 UDPPort

The UDPPort field of a data structure of type HBA_MGMTINFO shall contain the value of the UDP/TCP Port Number field of the Specific Identification Data in an RNID Accept with the Node Identification Data Format set to DFh (see FC-LS-2), indicating the management UDP/TCP port.

6.13.3.8 IPAddress

The IPAddress field of a data structure of type HBA_MGMTINFO shall contain the value of the IP address field of the Specific Identification Data in an RNID Accept with the Node Identification Data Format set to DFh (see FC-LS-2), indicating the management IP address.

The least significant byte of the IP address field of the RNID Specific Identification Data structure shall be stored in byte zero of the HBA_MGMTINFO IPAddress array, and successively higher order bytes of the IP address field of the RNID Specific Identification Data structure shall be stored in successively higher numbered bytes of the HBA_MGMTINFO IPAddress array.

6.13.3.9 TopologyDiscoveryFlags

The TopologyDiscoveryFlags field of a data structure of type HBA_MGMTINFO shall contain the value of the vendor specific field in word 12 of the Specific Identification Data in an RNID Accept with the Node Identification Data Format set to DFh (see FC-LS-2).

7 Function Calls

7.1 Overview

Requirements are specified in annex A for support of the functions specified in clause 7.

Table 6 is a list of the functions specified by this clause.

Table 6 — Function Summary and Requirements (part 1 of 3)

Function	Reference
Library Control Functions	
SMHBA2_GetVersion	7.2.1
HBA_LoadLibrary	7.2.2
HBA_FreeLibrary	7.2.3
SMHBA2_RegisterLibrary	7.2.4
SMHBA_GetWrapperLibraryAttributes	7.2.5
SMHBA_GetVendorLibraryAttributes	7.2.6
HBA_GetNumberOfAdapters	7.2.7
Object Attribute Functions	
SMHBA2_GetAdapterHandleByIndex	7.3.1
SMHBA2_GetAdapterAttributes	7.3.2
SMHBA2_GetAdapterBusAddress	7.3.4
SMHBA_GetNumberofPorts	7.3.3
SMHBA2_GetPortType	7.3.5
SMHBA2_GetPortAttributes	7.3.6
SMHBA2_GetPortAttributesByWWN	7.3.7
SMHBA2_GetPhyType	7.3.8
SMHBA2_GetPhyAttributes	7.3.9
SMHBA2_GetPhyCtlrAttributes	7.3.10
SMHBA2_GetFabricInfo	7.3.11
Object Relationship Functions	
SMHBA2_GetPortsOnAdapter	7.4.1

Table 6 — Function Summary and Requirements (part 2 of 3)

Function	Reference
SMHBA2_GetAdapterForPort	7.4.2
SMHBA2_GetLEPForPort	7.4.3
SMHBA2_GetDiscoveredPorts	7.4.4
SMHBA2_GetPhysOnAdapter	7.4.5
SMHBA2_GetAdapterForPhy	7.4.6
SMHBA2_GetPortsOnPhy	7.4.7
SMHBA2_GetPhysForPort	7.4.8
SMHBA2_GetCtlrForPhy	7.4.9
SMHBA2_GetPhyForCtlr	7.4.10
SMHBA2_GetFabricsForCtlr	7.4.11
SMHBA2_GetCtlrsForFabric	7.4.12
SMHBA2_GetFabricForPort	7.4.13
SMHBA2_GetPortsForFabric	7.4.14
Statistics Functions	
SMHBA2_GetPortStatistics	7.5.1
SMHBA2_GetProtocolStatistics	7.5.2
SMHBA2_GetPhyStatistics	7.5.3
SMHBA2_GetFIPStatistics	7.5.4
Target Information Functions	
SMHBA_GetBindingCapability	7.7.1
SMHBA_GetBindingSupport	7.7.2
SMHBA_SetBindingSupport	7.7.3
SMHBA_GetTargetMapping	7.7.4
SMHBA_GetPersistentBinding	7.7.5
SMHBA_SetPersistentBinding	7.7.6
SMHBA_RemovePersistentBindings	7.7.7
SMHBA_RemoveAllPersistentBindings	7.7.8

Table 6 — Function Summary and Requirements (part 3 of 3)

Function	Reference
SMHBA_GetLUNStatistics	7.7.9
SCSI Information Functions	
SMHBA_ScsiInquiry	7.8.1
SMHBA_ScsiReportLuns	7.8.2
SMHBA_ScsiReadCapacity	7.8.3
SMHBA_ScsiManagementIn	7.8.4
SMHBA_ScsiManagementOut	7.8.5
Fabric and Domain Management Functions	
HBA_SendCTPassThruV2	7.6.1
HBA_SetRNIDMgmtInfo	7.6.2
HBA_GetRNIDMgmtInfo	7.6.3
HBA_SendRNIDV2	7.6.4
HBA_SendSRL	7.6.5
HBA_SendLIRR	7.6.6
HBA_SendRLS	7.6.7
SMHBA_SendTEST	7.6.8
SMHBA_SendECHO	7.6.9
SMHBA_SendSMPPassThru	7.6.10
Event Handling Functions	
SMHBA_RegisterForAdapterAddEvents	7.9.2
SMHBA_RegisterForAdapterEvents	7.9.3
SMHBA_RegisterForAdapterPortEvents	7.9.4
SMHBA_RegisterForAdapterPortStatEvents	7.9.5
SMHBA2_RegisterForAdapterPhyStatEvents	7.9.6
SMHBA_RegisterForTargetEvents	7.9.7
HBA_RegisterForLinkEvents	7.9.8
HBA_RemoveCallback	7.9.9

7.2 Library Control Functions

7.2.1 SMHBA2_GetVersion

7.2.1.1 Format

```
HBA_UINT32 SMHBA2_GetVersion();
```

7.2.1.2 Description

The SMHBA2_GetVersion function shall return the version of the SM HBA API specification with which the SM HBA API library is compatible.

7.2.1.3 Arguments

None.

7.2.1.4 Return Values

The returned function value shall have a value that is a bit-wise OR of values to indicate the versions of the SM HBA API specification with which the library is compliant. The values shall be as specified in table 7.

Table 7 — Returned Function Values for SMHBA2_GetVersion

Value	Specification Version
1	FC-HBA
2	SM-HBA
4	This standard
any other	Reserved

NOTE 22 - It is suggested that Management Client Applications be capable of handling return values that are currently reserved. This would obviate the necessity to have a separate version function, e.g., HBA_GetVersion, SMHBA_GetVersion, for future revisions of this specification.

7.2.2 HBA_LoadLibrary

7.2.2.1 Format

```
HBA_STATUS HBA_LoadLibrary();
```

7.2.2.2 Description

The HBA_LoadLibrary function shall perform any initialization not inherent in the loading of an HBA API library by an application.

The HBA_LoadLibrary function in a wrapper library shall

a) perform common initialization;
b) determine the configured HBA specific libraries;
c) load the configured HBA specific libraries;
d) load the HBA specific libraries' function tables; and

e) call the HBA specific libraries' HBA_LoadLibrary functions.

If incompatibilities are detected among the wrapper library, its configured HBA specific libraries, and the drivers associated with the configured HBAs, any HBA specific libraries with which no incompatibility was detected shall be loaded.

The HBA_LoadLibrary function in an HBA specific library shall perform vendor specific initialization.

7.2.2.3 Arguments

None.

7.2.2.4 Return Values

The returned function value shall be as specified in table 8.

Table 8 — Returned Function Values for HBA_LoadLibrary

Value	Function result
HBA_STATUS_OK	The library and all its configured HBA specific libraries loaded properly.
HBA_STATUS_ERROR_ALREADY_LOADED	A library is already loaded.
HBA_STATUS_ERROR_INCOMPATIBLE	Incompatibilities were detected among the wrapper library, its configured HBA specific libraries, and the drivers associated with the configured HBAs.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.2.3 HBA_FreeLibrary

7.2.3.1 Format

```
HBA_STATUS HBA_FreeLibrary();
```

7.2.3.2 Description

The HBA_FreeLibrary function shall free the system resources used by the called library. It shall be called after all HBA library functions are complete to free all resources.

7.2.3.3 Arguments

None.

7.2.3.4 Return Values

The returned function value shall be as specified in table 9.

Table 9 — Returned Function Values for HBA_FreeLibrary

Value	Function result
HBA_STATUS_OK	All resources used by the library and all its configured HBA specific libraries were freed properly.
HBA_STATUS_ERROR_NOT_LOADED	No library is currently loaded.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.2.4 SMHBA2_RegisterLibrary

7.2.4.1 Format

```
HBA_UINT32 SMHB2A_RegisterLibrary(SMHBA2_ENTRYPOINTS *pSMHBA2Info);
```

7.2.4.2 Description

The SMHBA2_RegisterLibrary function shall register the SM-HBA-2 specific library functions with the wrapper library. This shall be implemented by a SM-HBA-2 specific library and called by the wrapper library. The SMHBA2_RegisterLibrary function shall not register those library functions that are required only by previous revisions of this standard.

7.2.4.3 Arguments

Argument **pSMHBA2Info** shall be a pointer to a structure in which the entry addresses of the vendor specific implementation of all library functions may be returned.

7.2.4.4 Return Values

The returned function value shall be as specified in table 10

Table 10 — Returned Function Values for SMHBA2_RegisterLibrary

Value	Function result
HBA_STATUS_OK	Function pointers have been returned.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.2.5 SMHBA_GetWrapperLibraryAttributes

7.2.5.1 Format

```
HBA_UINT32 SMHBA_GetWrapperLibraryAttributes(
       SMHBA_LIBRARYATTRIBUTES *attributes
);
```

7.2.5.2 Description

The SMHBA_GetWrapperLibraryAttributes function shall return details about the implementation of the wrapper library in which the call is implemented (e.g., allowing software to determine whether a compatible library is installed, and allowing installation software to describe to an operator a library to be replaced).

In an SM HBA library with OS specific structure, the SMHBA_GetWrapperLibraryAttributes function returns information about OS specific software that presents the API.

7.2.5.3 Arguments

Argument **attributes** shall be a pointer to a structure in which the attributes of the library implementing the call may be returned.

7.2.5.4 Return Values

The returned function value shall have a value that is a bit-wise OR of values to indicate the versions of the SM HBA API specification with which the library is compliant. The values shall be as specified in table 11.

Table 11 — Returned Function Values for SMHBA_GetWrapperLibraryAttributes

Value	Specification Version
1	SM-HBA
2	This standard
any other	Reserved

The structure pointed to by **attributes** shall contain the attributes of the specified SM HBA specific library. If it is not practical to determine the LibPath attribute, a null string may be returned for that attribute.

7.2.6 SMHBA_GetVendorLibraryAttributes

7.2.6.1 Format

```
HBA_UINT32 SMHBA_GetVendorLibraryAttributes(
       HBA_UINT32             adapter_index;
       SMHBA_LIBRARYATTRIBUTES *attributes;
);
```

7.2.6.2 Description

The SMHBA_GetVendorLibraryAttributes function shall return details about the implementation of the HBA specific library associated with the specified HBA (e.g., allowing software including a wrapper library, to determine whether a compatible library is installed, and allowing installation software to describe to an operator a library about to be replaced).

An HBA specific library shall ignore the adapter_index parameter.

In an HBA API with OS specific structure, the SMHBA_GetVendorLibraryAttributes function returns information about the OS specific software that presents the API. This shall be the same as the information returned by SMHBA_GetWrapperLibraryAttributes (see 7.2.5).

7.2.6.3 Arguments

Argument **adapter_index** shall be an index to an HBA in the range of the return value of HBA_GetNumberofAdapters (see 7.2.7). The version details shall be returned for the SM HBA specific library that interfaces to the indexed HBA.

In an SM HBA API library with OS specific structure, the SMHBA_GetVendorLibraryAttributes function returns information about the OS specific software that presents the API regardless of the HBA that is indexed.

More than one HBA may be interfaced by the same library, so more than one index may cause the same set of library details to be returned.

Argument **attributes** shall be a pointer to a structure in which the attributes of the library implementing the call may be returned.

7.2.6.4 Return Values

The returned function value shall have a value that is a bit-wise OR of values to indicate the versions of the SM HBA API specification with which the library is compliant. The values shall be as specified in table 12.

Table 12 — Returned Function Values for SMHBA_GetVendorLibraryAttributes

Value	Specification Version
1	SM-HBA
2	This standard
any other	Reserved

The structure pointed to by **attributes** shall contain the attributes of the specified SM HBA specific library. If it is not practical to determine the LibPath attribute, a null string may be returned for that attribute.

7.2.7 HBA_GetNumberOfAdapters

7.2.7.1 Format

```
HBA_UINT32 HBA_GetNumberOfAdapters();
```

7.2.7.2 Description

The HBA_GetNumberOfAdapters function shall return the number of HBAs supported by the library. This shall be the current number of HBAs. The value returned shall reflect dynamic change of HBA inventory without requiring restart of the system, driver, or library.

7.2.7.3 Arguments

None.

7.2.7.4 Return Values

The returned function value shall be the number of HBAs supported by this library. If no HBAs are supported, the library shall return zero.

7.3 Object Attribute Functions

7.3.1 SMHBA2_GetAdapterHandleByIndex

7.3.1.1 Format

```
HBA_STATUS HBA_GetAdapterHandleByIndex(
       HBA_UINT32 adapterindex,
       HBA_HANDLE *pAdapterhandle
);
```

7.3.1.2 Description

The SMHBA2_GetAdapterHandleByIndex function shall return the HBA_HANDLE for the indexed HBA.

7.3.1.3 Arguments

Argument **adapterindex** shall be the index of the HBA for which a handle is to be returned, among all the adapters among all the adapters supported by the library. Its minimum value is zero. Its maximum value is one less than the value returned by HBA_GetNumberOfAdapters.

Argument **pAdapterhandle** shall be a pointer to space in which the HBA handle may be returned.

7.3.1.4 Return Values

The returned function value shall be as specified in table 13.

Table 13 — Returned Function Values for SMHBA2_GetAdapterHandleByIndex

Value	Function result
HBA_STATUS_OK	An adapter handle has been returned.
HBA_STATUS_ERROR_ILLEGAL_INDEX	There is no HBA with the adapter index identified by **adapterindex**.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the value in the buffer pointed to by **pAdapterhandle** shall be an adapter handle for the adapter indexed by **adapterindex.** If the returned function value is not HBA_STATUS_OK, the value in the buffer pointed to by **pAdapterhandle** shall be zero.

For HBA API libraries with OS independent structure (i.e., a wrapper library and HBA specific libraries), the high order 16 bits of the value shall be zero when returned by an HBA specific library. The high order 16 bits of the value shall be assigned by a wrapper library to uniquely identify the HBA specific library that handles the HBA indicated.

7.3.2 SMHBA2_GetAdapterAttributes

7.3.2.1 Format

```
HBA_STATUS SMHBA2_GetAdapterAttributes(
       HBA_HANDLE handle,
       SMHBA2_ADAPTERATTRIBUTES *pAdapterAttributes
);
```

7.3.2.2 Description

The SMHBA2_GetAdapterAttributes function shall return the attributes for an HBA.

7.3.2.3 Arguments

Argument **handle** shall be an HBA_HANDLE to the HBA for which attributes are requested.

Argument **pAdapterAttribute** shall be a pointer to a structure in which attributes for the HBA may be returned.

7.3.2.4 Return Values

The returned function value shall be as specified in table 14 .

Table 14 — Returned Function Values for SMHBA2_GetAdapterAttributes

Value	Function Result
HBA_STATUS_OK	HBA attributes have been returned.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.3.3 SMHBA_GetNumberofPorts

7.3.3.1 Format

```
HBA_STATUS SMHBA_GetNumberofPorts(
       HBA_HANDLE handle,
       HBA_UINT32 *numberofports
);
```

7.3.3.2 Description

The SMHBA_GetNumberofPorts() function shall return the total number of SAS and/or FC ports available through the given HBA. This shall be the current number of ports, and it reflects the dynamic change of ports inventory without requiring the restart of the system, driver or library.

NOTE 23 - The number of ports reported by an adapter that supports SAS may vary depending on the number of SAS domains to which its phys are attached. More than one SAS Port may have the same Port_Identifier if they are all in different SAS domains.

7.3.3.3 Arguments

Argument **handle** shall be an HBA_HANDLE to an HBA for which the number of ports are requested.

Argument **numberofports** shall be a pointer to an integer in which the total number of ports available through this HBA may be returned. A value of zero indicates that this HBA currently does not have any ports available.

7.3.3.4 Return Values

The returned function value shall be as specified in table 15.

Table 15 — Returned Function Values for SMHBA_GetNumberofPorts

Value	Function Result
HBA_STATUS_OK	Number of ports has been returned.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.3.4 SMHBA2_GetAdapterBusAttributes

7.3.4.1 Format

```
HBA_STATUS SMHBA2_GetAdapterBusAttributes(
       HBA_HANDLE handle,
       SMHBA2_BUSADDRESS *pAdapterBusAddress
);
```

7.3.4.2 Description

The SMHBA2_GetAdapterBusAttributes function shall return the bus attributes for an HBA.

7.3.4.3 Arguments

Argument **handle** shall be an HBA_HANDLE to the HBA for which bus attributes are requested.

Argument **pAdapterBusAddress** shall be a pointer to a structure in which attributes for the adapter bus may be returned.

7.3.4.4 Return Values

The returned function value shall be as specified in table 14 .

Table 16 — Returned Function Values for SMHBA2_GetAdapterBusAttributes

Value	Function Result
HBA_STATUS_OK	Adapter bus attributes have been returned.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.3.5 SMHBA2_GetPortType

7.3.5.1 Format

```
HBA_STATUS SMHBA2_GetPortType(
       HBA_HANDLE portHandle,
       SMHBA2_PORTTYPE *porttype
);
```

7.3.5.2 Description

The SMHBA2_GetPortType function shall retreive the port type attribute for a specified end port on an HBA.

7.3.5.3 Arguments

Argument **portHandle** shall be an HBA_HANDLE to a port whose type is being requested.

Argument **porttype** shall be a pointer to a structure in which the type of the specified end port may be returned.

7.3.5.4 Return Values

The returned function value shall be as specified in table 17

Table 17 — Returned Function Values for SMHBA2_GetPortType

Value	Function result
HBA_STATUS_OK	Port type has been returned.
HBA_STATUS_ERROR_NOT_SUPPORTED	The port referenced by **portHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.3.6 SMHBA2_GetPortAttributes

7.3.6.1 Format

```
HBA_STATUS SMHBA2_GetPortAttributes(
       HBA_HANDLE portHandle,
       SMHBA2_PORTATTRIBUTES *portattributes
);
```

7.3.6.2 Description

The SMHBA2_GetPortAttributes function shall return the port type specific attributes for a local end port or port discovered in the network or domain.

7.3.6.3 Arguments

Argument **portHandle** shall be an HBA_HANDLE to a port whose attributes are being requested.

Argument **portattributes** shall be a pointer to a structure in which the port attributes specific to the local end port type or port discovered in the network or domain may be returned.

NOTE 24 - The management client application is assumed to have prior knowledge of the port type attribute of the end port whose attributes are being retrieved. In addition, the management client application shall also allocate appropriate memory to hold the attributes of the end port prior to invoking SMHBA2_GetPortAttributes.

7.3.6.4 Return Values

The returned function value shall be as specified in table 18.

Table 18 — Returned Function Values for SMHBA2_GetPortAttributes

Value	Function result
HBA_STATUS_OK	Port attributes have been returned.
HBA_STATUS_ERROR_NOT_SUPPORTED	The port referenced by **portHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.3.7 SMHBA2_GetPortAttributesByWWN

7.3.7.1 Format

```
HBA_STATUS SMHBA2_GetPortAttributesByWWN(
       HBA_HANDLE    portHandle,
       HBA_WWN       portWWN,
       HBA_WWN       domainPortWWN,
       SMHBA2_PORTATTRIBUTES *portattributes
);
```

7.3.7.2 Description

The SMHBA2_GetPortAttributesByWWN function shall return the port type specific attributes for a local end port or discovered end port as specified by the portWWN identifier.

7.3.7.3 Arguments

Argument **handle** shall be an HBA_HANDLE to an HBA through which the attributes of a port are being queried.

Argument **portWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port to query.

Argument **portattributes** shall be a pointer to a structure in which the port attributes specific to the port type may be returned.

NOTE 25 - The management client application is assumed to have prior knowledge of the port type attribute of the end port whose attributes are being retrieved. In addition, the management client application shall also allocate appropriate memory to hold the attributes of the end port prior to invoking SMHBA_GetPortAttributesByWWN.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP port discovered through a local port on the specified HBA or zero. If domainPortWWN is not zero, the port to query shall be attached to the same SAS domain as the specified SMP port. Argument domainPortWWN shall be ignored if the specified port is not a SAS Port.

7.3.7.4 Return Values

The returned function value shall be as specified in table 19.

Table 19 — Returned Function Values for SMHBA2_GetPortAttributesByWWN

Value	Function result
HBA_STATUS_OK	Port attributes have been returned.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local end port or discovered FC_Port with Name_Identifier identified by **portWWN**, or the HBA identified by **handle** is not able to access a local end port or discovered SAS Port with Port_Identifier identified by **portWWN,** or the HBA identified by **handle** is not able to access an expander SMP target port by **domainPortWWN**.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.3.8 SMHBA2_GetPhyType

7.3.8.1 Format

```
HBA_STATUS SMHBA2_GetPhyType(
       HBA_HANDLE phyHandle,
       SMHBA2_PHYTYPE *pPhyType
);
```

7.3.8.2 Description

The SMHBA2_GetPhyType function shall return the phy type attribute of the specified phy.

7.3.8.3 Arguments

Argument **phyHandle** shall be an HBA_HANDLE to the phy.

Argument **pPhyType** is a pointer to space in which the phy type may be returned.

7.3.8.4 Return Values

The returned function value shall be as specified in table 20.

Table 20 — Returned Function Values for SMHBA2_GetPhyType

Value	Function result
HBA_STATUS_OK	Phy type has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**phyHandle** is not a phy HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The phy referenced by **phyHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.3.9 SMHBA2_GetPhyAttributes

7.3.9.1 Format

```
HBA_STATUS SMHBA2_GetPhyAttributes(
       HBA_HANDLE phyHandle,
       SMHBA2_PHYATTRIBUTES *pPhyAttributes
);
```

7.3.9.2 Description

The SMHBA2_GetPhyAttributes function shall return the attributes of the phy.

7.3.9.3 Arguments

Argument **phyHandle** shall be an HBA_HANDLE to the phy.

Argument **pPhyAttributes** is a pointer to a structure in which attributes for the phy may be returned.

7.3.9.4 Return Values

The returned function value shall be as specified in table 21.

Table 21 — Returned Function Values for SMHBA2_GetPhyAttributes

Value	Function result
HBA_STATUS_OK	Phy attributes have been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**phyHandle** is not a PHY HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The PHY referenced by **phyHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.3.10 SMHBA2_GetPhyCtlrAttributes

7.3.10.1 Format

```
HBA_STATUS SMHBA2_GetPhyCtlrAttributes(
       HBA_HANDLE ctlrHandle,
       SMHBA2_NCPATTRIBUTES *pNPCAttributes
);
```

7.3.10.2 Description

The SMHBA2_GetPhyCtlrAttributes function shall return the attributes of an n_port controller.

7.3.10.3 Arguments

Argument **ctlrHandle** shall be an HBA_HANDLE to the n_port controller.

Argument **pNPCAttributes** is a pointer to a structure in which attributes for the n_port controller may be returned.

7.3.10.4 Return Values

The returned function value shall be as specified in table 22.

Table 22 — Returned Function Values for SMHBA2_GetPhyCtlrAttributes

Value	Function result
HBA_STATUS_OK	N_port Controller attributes have been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**ctlrHandle** is not a n_port controller HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The n_port controller referenced by **cltrHandle** does not support this function.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some mapping entries may have been returned.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned value is HBA_STATUS_OK OR HBA_STATUS_ERROR_MORE_DATA AND the controller type is SMHBA2_NPCTYPE_FCOE, the structure pointed to by DiscoveredFCFMACAddrList in the NPCSpecificAttributes of **pNPCAttributes** shall contain pointers to SMHBA2_DiscoveredFCFMACAddrs for this FCoE controller. The number of entries in the structure shall be the minimum of the number of entries specified at the function call or the full mapping. The value of the NumberOfEntries field of the returned structure DiscoveredFCFMACAddrList shall be the total number of Discovered FCF MAC Address structures even when the function returns an error because the buffer is too small to return all of the Discovered FCF MAC Address structures. An application may either allocate a sufficiently large buffer or check this value after a read, or do a read of the NumberOfEntries values separately and allocate a new buffer given the size to accommodate the entire list.

7.3.11 SMHBA2_GetFabricInfo

7.3.11.1 Format

```
HBA_STATUS SMHBA2_GetFabricInfo(
       HBA_HANDLE fabricHandle,
       SMHBA2_FABRICINFO pFabricInfo
);
```

7.3.11.2 Description

The SMHBA2_GetFabricInfo function shall return the attributes of a fabric.

7.3.11.3 Arguments

Argument **fabric Handle** shall be an HBA_HANDLE to the fabric.

Argument **pFabricInfo** is a pointer to a structure in which the attributes for the fabric may be returned.

7.3.11.4 Return Values

The returned function value shall be as specified in table 23.

Table 23 — Returned Function Values for SMHBA2_GetFabricInfo

Value	Function result
HBA_STATUS_OK	Fabric attributes have been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**fabricHandle** is not a fabric HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The fabric referenced by **fabricHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.4 Object Relationship functions

7.4.1 SMHBA2_GetPortsOnAdapter

7.4.1.1 Format

```
HBA_STATUS SMHBA2_GetPortsOnAdapter(
       HBA_HANDLE handle,
       SMHBA_HANDLELIST *pPortHandleList
);
```

7.4.1.2 Description

The SMHBA2_GetPortsOnAdapter function shall return a list of local end FC or SAS port handles for an HBA.

7.4.1.3 Arguments

Argument **handle** shall be an HBA_HANDLE of an HBA.

Argument **pPortHandleList** shall be a pointer to a structure in which a list of ports for the specified HBA may be returned. For FC, all ports shall be of type VN_Port. The size of this structure shall be limited by the NumberOfEntries value within the structure.

7.4.1.4 Return Values

The retuned function value shall be as specified in table 24.

Table 24 — Returned Function Values for SMHBA2_GetPortsOnAdapter

Value	Function result
HBA_STATUS_OK	A list of ports has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**handle** is not a HBA HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some mapping entries may have been returned.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned value is HBA_STATUS_OK OR HBA_STATUS_ERROR_MORE_DATA, the structure pointed to by **pPortHandleList** shall contain HBA_HANDLEs of local end FC or SAS port handles for an HBA. The number of entries in the structure shall be the minimum of the number of entries specified at the function call or the full mapping. The value of the NumberOfEntries field of the returned **pPortHandleList** structure shall be the total number of local FC or SAS port handles for an HBA even when the function returns an error because the buffer is too small to return all of the local FC or SAS ports for the HBA. An application may either allocate a sufficiently large buffer or check this value after a read, or do a read of the NumberOfEntries values separately and allocate a new buffer given the size to accommodate the entire list.

7.4.2 SMHBA2_GetAdapterForPort

7.4.2.1 Format

```
HBA_STATUS SMHBA2_GetAdapterForPort(
       HBA_HANDLE portHandle,
       HBA_HANDLE *pAdapterHandle
);
```

7.4.2.2 Description

The SMHBA2_GetAdapterForPort function shall return the handle for the HBA of a local end port.

7.4.2.3 Arguments

Argument **portHandle** shall be an HBA_HANDLE of a local end port.

Argument **pAdapterHandle** shall be a pointer to the space in which the HBA handle may be returned.

7.4.2.4 Return Values

The returned function value shall be as specified in table 25.

Table 25 — Returned Function Values for SMHBA2_GetAdapterForPort

Value	Function result
HBA_STATUS_OK	Adapter handle has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**portHandle** is not a port HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The port referenced by **portHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.4.3 SMHBA2_GetLEPForPort

7.4.3.1 Format

```
HBA_STATUS SMHBA2_GetLEPForPort(
       HBA_HANDLE portHandle,
       SMHBA2_FCOE_LEP *pFCoELEP
);
```

7.4.3.2 Description

The SMHBA2_GetLEPForPort function shall return the Local Endpoint addresses for an FCoE port.

7.4.3.3 Arguments

Argument **portHandle** shall be an HBA_HANDLE of a FCoE port.

Argument **pFCoELEP** shall be a pointer to the structure in which the Link Endpoint addresses for the port may be returned.

7.4.3.4 Return Values

The returned function value shall be as specified in table 26.

Table 26 — Returned Function Values for SMHBA2_GetLEPForPort

Value	Function result
HBA_STATUS_OK	Link Endpoint addresses have been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**portHandle** is not a FCoE port HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The port referenced by **portHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.4.4 SMHBA2_GetDiscoveredPorts

7.4.4.1 Format

```
HBA_STATUS SMHBA2_GetDiscoveredPorts(
       HBA_HANDLE portHandle,
       SMHBA_HANDLELIST *pPortHandleList
);
```

7.4.4.2 Description

The SMHBA2_GetDiscoveredPorts function shall return a list of port handles for ports discovered through the specified port.

7.4.4.3 Arguments

Argument **portHandle** shall be an HBA_HANDLE of a local end port.

Argument **pPortHandleList** shall be a pointer to a structure in which a list of ports discovered through the specified port may be returned. If the specified port is not local, the list may be empty.

7.4.4.4 Return Values

The returned function value shall be as specified in table 27.

Table 27 — Returned Function Values for SMHBA2_GetDiscoveredPorts

Value	Function result
HBA_STATUS_OK	A list of discovered ports has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**portHandle** is not a local end port HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The port referenced by **portHandle** does not support this function.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some mapping entries may have been returned.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned value is HBA_STATUS_OK OR HBA_STATUS_ERROR_MORE_DATA, the structure pointed to by **pPortHandleList** shall contain HBA_HANDLEs of ports discovered through the specified port. The number of entries in the structure shall be the minimum of the number of entries specified at the function call or the full mapping. The value of the NumberOfEntries field of the returned **pPortHandleList** structure shall be the total number of discovered port handles for a port even when the function returns an error because the buffer is too small to return all of the discovered ports for the local port. An application may either allocate a sufficiently large buffer or check this value after a read, or do a read of the NumberOfEntries values separately and allocate a new buffer given the size to accommodate the entire list.

7.4.5 SMHBA2_GetPhysOnAdapter

7.4.5.1 Format

```
HBA_STATUS SMHBA2_GetPhysOnAdapter(
       HBA_HANDLE handle,
       SMHBA_HANDLELIST *pPhyHandleList
);
```

7.4.5.2 Description

The SMHBA2_GetPhysOnAdapter function shall return a list phy handles for the given HBA.

7.4.5.3 Arguments

Argument **handle** shall be an HBA_HANDLE of an HBA.

Argument **pPhyHandleList** shall be a pointer to a structure in which a list of phys for the HBA.

7.4.5.4 Return Values

The returned function value shall be as specified in table 28.

Table 28 — Returned Function Values for SMHBA2_GetPhysOnAdapter

Value	Function result
HBA_STATUS_OK	A list of phys has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**handle** is not a HBA HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some mapping entries may have been returned.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned value is HBA_STATUS_OK OR HBA_STATUS_ERROR_MORE_DATA, the structure pointed to by **pPhyHandleList** shall contain HBA_HANDLEs of phys for the HBA. The number of entries in the structure shall be the minimum of the number of entries specified at the function call or the full mapping. The value of the NumberOfEntries field of the returned **pPhyHandleList** structure shall be the total number of phys for the HBA even when the function returns an error because the buffer is too small to return all of the phy HANDLEs. An application may either allocate a sufficiently large buffer or check this value after a read, or do a read of the NumberOfEntries values separately and allocate a new buffer given the size to accommodate the entire list.

7.4.6 SMHBA2_GetAdapterForPhy

7.4.6.1 Format

```
HBA_STATUS SMHBA2_GetAdapterForPhy(
       HBA_HANDLE phyHandle,
       HBA_HANDLE *pAdapterHandle
);
```

7.4.6.2 Description

The SMHBA2_GetAdapterForPhy function shall return a handle of the HBA of the specified phy.

7.4.6.3 Arguments

Argument **phyHandle** shall be an HBA_HANDLE of a phy.

Argument **pAdapterHandle** shall be a pointer to the space in which the HBA handle may be returned.

7.4.6.4 Return Values

The returned function value shall be as specified in table 29.

Table 29 — Returned Function Values for SMHBA2_GetAdapterForPhy

Value	Function result
HBA_STATUS_OK	An HBA handle has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**phyHandle** is not a phy HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The phy referenced by **phyHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.4.7 SMHBA2_GetPortsOnPhy

7.4.7.1 Format

```
HBA_STATUS SMHBA2_GetPortsOnPhy(
       HBA_HANDLE phyHandle,
       SMHBA_HANDLELIST *pPortHandleList
);
```

7.4.7.2 Description

The SMHBA2_GetPortsOnPhy function shall return a list of local end FC or SAS port handles for the given phy.

7.4.7.3 Arguments

Argument **phyHandle** shall be an HBA_HANDLE of a phy.

Argument **pPortHandleList** shall be a pointer to a structure in which a list of ports on the specified phy may be returned.

7.4.7.4 Return Values

The returned function value shall be as specified in table 30.

Table 30 — Returned Function Values for SMHBA2_GetPortsOnPhy

Value	Function result
HBA_STATUS_OK	A list of port handles has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**phyHandle** is not a phy HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The phy referenced by **phyHandle** does not support this function.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some mapping entries may have been returned.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned value is HBA_STATUS_OK OR HBA_STATUS_ERROR_MORE_DATA, the structure pointed to by **pPortHandleList** shall contain HBA_HANDLEs of ports for the phy. The number of entries in the structure shall be the minimum of the number of entries specified at the function call or the full mapping. The value of the NumberOfEntries field of the returned **pPortHandleList** structure shall be the total number of ports for the phy even when the function returns an error because the buffer is too small to return all of the port HANDLEs. An application may either allocate a sufficiently large buffer or check this value after a read, or do a read of the NumberOfEntries values separately and allocate a new buffer given the size to accommodate the entire list.

7.4.8 SMHBA2_GetPhysForPort

7.4.8.1 Format

```
HBA_STATUS SMHBA2_GetPhysForPort(
       HBA_HANDLE portHandle,
       SMHBA_HANDLELIST *pPhyHandleList
);
```

7.4.8.2 Description

The SMHBA2_GetAdapterForPort function shall return a list of phy handles for the specified local end port.

7.4.8.3 Arguments

Argument **portHandle** shall be an HBA_HANDLE of a local end port.

Argument **pPhyHandleList** shall be a pointer to a structure in which a list of phys on the specified port may be returned. For an FC port, one phy handle will be returned

7.4.8.4 Return Values

The returned function value shall be as specified in table 31.

Table 31 — Returned Function Values for SMHBA2_GetPhysForPort

Value	Function result
HBA_STATUS_OK	A list of phy handles has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**portHandle** is not a port HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The port referenced by **portHandle** does not support this function.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some mapping entries may have been returned.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned value is HBA_STATUS_OK OR HBA_STATUS_ERROR_MORE_DATA, the structure pointed to by **pPhyHandleList** shall contain HBA_HANDLEs of phys for a port. The number of entries in the structure shall be the minimum of the number of entries specified at the function call or the full mapping. The value of the NumberOfEntries field of the returned **pPhyHandleList** structure shall be the total number of phy handles for the port even when the function returns an error because the buffer is too small to return phys for the port. An application may either allocate a sufficiently large buffer or check this value after a read, or do a read of the NumberOfEntries values separately and allocate a new buffer given the size to accommodate the entire list.

7.4.9 SMHBA2_GetCtlrForPhy

7.4.9.1 Format

```
HBA_STATUS SMHBA2_GetCtlrForPhy(
       HBA_HANDLE phyHandle,
       HBA_HANDLE *pCtlrHandle
);
```

7.4.9.2 Description

The SMHBA2_GetCtlrForPhy function shall return the n_port controller handle for the given phy.

7.4.9.3 Arguments

Argument **handle** shall be an HBA_HANDLE of a phy.

Argument **pCtlrHandle** shall be a pointer to the space in which the handle of a n_port Controller for the specified phy may be returned.

7.4.9.4 Return Values

The returned function value shall be as specified in table 32.

Table 32 — Returned Function Values for SMHBA2_GetCtlrForPhy

Value	Function result
HBA_STATUS_OK	An n_port controller handle has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**phyHandle** is not a phy HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The phy referenced by **phyHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.4.10 SMHBA2_GetPhyForCtlr

7.4.10.1 Format

```
HBA_STATUS SMHBA2_GetPhyForCtlr(
       HBA_HANDLE ctlrHandle,
       HBA_HANDLE *pPhyHandle
);
```

7.4.10.2 Description

The SMHBA2_GetPhyForCtlr function shall return the phy handle for the specified n_port controller.

7.4.10.3 Arguments

Argument **ctlrHandle** shall be an HBA_HANDLE of a n_port controller.

Argument **pPhyHandle** shall be a pointer to the space in which the phy handle may be returned.

7.4.10.4 Return Values

The returned function value shall be as specified in table 33.

Table 33 — Returned Function Values for SMHBA2_GetPhyForCtlr

Value	Function result
HBA_STATUS_OK	An phy handle has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**ctlrHandle** is not an n_port controller HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The n_port controller referenced by **ctlrHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.4.11 SMHBA2_GetFabricsForCtlr

7.4.11.1 Format

```
HBA_STATUS SMHBA2_GetFabricsForCtlr(
       HBA_HANDLE ctlrHandle,
       SMHBA_HANDLELIST *pFabricHandleList
);
```

7.4.11.2 Description

The SMHBA2_GetFabricsForCtlr function shall return a list of fabric handles for the given phy.

7.4.11.3 Arguments

Argument **ctlrHandle** shall be an HBA_HANDLE of an n_port controller.

Argument **pFabricHandleList** shall be a pointer to a structure in which a list of fabric handles for the specified n_port controller may be returned.

7.4.11.4 Return Values

The returned function value shall be as specified in table 34.

Table 34 — Returned Function Values for SMHBA2_GetFabricsForCtlr

Value	Function result
HBA_STATUS_OK	A list of fabric handles has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**ctlrHandle** is not an n_port controller HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The n_port controller referenced by **ctlrHandle** does not support this function.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some mapping entries may have been returned.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned value is HBA_STATUS_OK OR HBA_STATUS_ERROR_MORE_DATA, the structure pointed to by **pFabricHandleList** shall contain HBA_HANDLEs of fabrics for an n_port controller. The number of entries in the structure shall be the minimum of the number of entries specified at the function call or the full mapping. The value of the NumberOfEntries field of the returned **pFabricHandleList** structure shall be the total number of fabrics for an n_port controller even when the function returns an error because the buffer is too small to return all fabric handles for an n_port controller. An application may either allocate a sufficiently large buffer or check this value after a read, or do a read of the NumberOfEntries values separately and allocate a new buffer given the size to accommodate the entire list.

7.4.12 SMHBA2_GetCtlrsForFabrics

7.4.12.1 Format

```
HBA_STATUS SMHBA2_GetCtlrsForFabrics(
       HBA_HANDLE fabricHandle,
       SMHBA_HANDLELIST *pCtlrHandleList
);
```

7.4.12.2 Description

The SMHBA2_GetFabricsForCtlr function shall return a list of n_port controller handles for a fabric.

7.4.12.3 Arguments

Argument **fabrichandle** shall be an HBA_HANDLE of an fabric.

Argument **pCtlrHandleList** shall be a pointer to a structure in which a list of n_port controller handles for the specified fabric may be returned.

7.4.12.4 Return Values

The returned function value shall be as specified in table 35.

Table 35 — Returned Function Values for SMHBA2_GetCtlrsForFabric

Value	Function result
HBA_STATUS_OK	A list of n_port controller handles has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**fabricHandle** is not an fabric HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The fabric referenced by **fabricHandle** does not support this function.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some mapping entries may have been returned.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned value is HBA_STATUS_OK OR HBA_STATUS_ERROR_MORE_DATA, the structure pointed to by **pCtlrHandleList** shall contain HBA_HANDLEs of n_port controllers for the fabric. The number of entries in the structure shall be the minimum of the number of entries specified at the function call or the full mapping. The value of the NumberOfEntries field of the returned **pCtlrHandleList** structure shall be the total number of n_port controller handles for a fabric even when the function returns an error because the buffer is too small to return all n_port controller handles for a fabric. An application may either allocate a sufficiently large buffer or check this value after a read, or do a read of the NumberOfEntries values separately and allocate a new buffer given the size to accommodate the entire list.

7.4.13 SMHBA2_GetFabricForPort

7.4.13.1 Format

```
HBA_STATUS SMHBA2_GetFabricForPort(
       HBA_HANDLE portHandle,
       HBA_HANDLE *pFabricHandle
);
```

7.4.13.2 Description

The SMHBA2_GetFabricForPort function shall return a fabric handle for the given port.

7.4.13.3 Arguments

Argument **porthandle** shall be an HBA_HANDLE of an port.

Argument **pFabricHandle** shall be a pointer to the space in which a fabric handle for the specified port may be returned.

7.4.13.4 Return Values

The returned function value shall be as specified in table 36.

Table 36 — Returned Function Values for SMHBA2_GetFabricForPort

Value	Function result
HBA_STATUS_OK	A list of fabric handles has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**portHandle** is not an port HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The port referenced by **portHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.4.14 SMHBA2_GetPortsForFabric

7.4.14.1 Format

```
HBA_STATUS SMHBA2_GetPortsForFabric(
       HBA_HANDLE fabricHandle,
       SMHBA_HANDLELIST *pPortHandleList
);
```

7.4.14.2 Description

The SMHBA2_GetPortsForFabric function shall return a list of port handles for the specified fabric.

7.4.14.3 Arguments

Argument **fabricHandle** shall be an HBA_HANDLE of a fabric.

Argument **pPortHandleList** shall be a pointer to a list port handles for the specified fabric may be returned.

7.4.14.4 Return Values

The returned function value shall be as specified in table 37.

Table 37 — Returned Function Values for SMHBA2_GetPortsForFabric

Value	Function result
HBA_STATUS_OK	A list of port handles has been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**fabricHandle** is not an fabric HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The fabric referenced by **fabricHandle** does not support this function.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some mapping entries may have been returned.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned value is HBA_STATUS_OK OR HBA_STATUS_ERROR_MORE_DATA, the structure pointed to by **pPortHandleList** shall contain HBA_HANDLEs of ports for the fabric. The number of entries in the structure shall be the minimum of the number of entries specified at the function call or the full mapping. The value of the NumberOfEntries field of the returned **pPortHandleList** structure shall be the total number of port handles for a fabric even when the function returns an error because the buffer is too small to return all port handles for a fabric. An application may either allocate a sufficiently large buffer or check this value after a read, or do a read of the NumberOfEntries values separately and allocate a new buffer given the size to accommodate the entire list.

7.5 Statistics Functions

7.5.1 SMHBA2_GetPortStatistics

7.5.1.1 Format

```
HBA_STATUS SMHBA2_GetPortStatistics(
       HBA_HANDLE portHandle,
       SMHBA_PORTSTATISTICS *pPortStatistics
);
```

7.5.1.2 Description

The SMHBA2_GetPortStatistics shall return the traffic statistics for a specified local end port.

7.5.1.3 Arguments

Argument **portHandle** shall be an HBA_HANDLE of a local end port.

Argument **pPortStatistics** shall be a pointer to a structure in which the statistics for the specified port may be returned.

7.5.1.4 Return Values

The returned function value shall be as specified in table 38.

Table 38 — Returned Function Values for SMHBA2_GetPortStatistics

Value	Function result
HBA_STATUS_OK	Protocol statistics have been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**portHandle** is not a port HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The port referenced by **portHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.5.2 SMHBA2_GetProtocolStatistics

7.5.2.1 Format

```
HBA_STATUS SMHBA2_GetProtocolStatistics(
       HBA_HANDLE portHandle,
       HBA_UINT32 protocolType,
       SMHBA_PROTOCOLSTATISTICS *pProtocolStatistics
);
```

7.5.2.2 Description

The SMHBA2_GetProtocolStatistics shall return the traffic statistics for a specified protocol via a specific local end port.

7.5.2.3 Arguments

Argument **portHandle** shall be an HBA_HANDLE to a local end port.

Argument **protocolType** shall be the FC-4 protocol type if the local end port is SMHBA_FC_Port.
Argument **protocolType** shall be the SAS protocol type if the local end port is a SAS Port.

Argument **pProtocolStatistics** shall be a pointer to a structure in which the statistics for the specified protocol may be returned.

7.5.2.4 Return Values

The returned function value shall be as specified in table 39.

Table 39 — Returned Function Values for SMHBA2_GetProtocolStatistics

Value	Function result
HBA_STATUS_OK	Protocol statistics shave been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**portHandle** is not a port HBA_HANDLE.
HBA_STATUS_ERROR_INVALID_PROTOCOL_TYPE	The protocol referenced by **protocoltype** is not valid.
HBA_STATUS_ERROR_NOT_SUPPORTED	The port referenced by **portHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.5.3 SMHBA2_GetPhyStatistics

7.5.3.1 Format

```
HBA_STATUS SMHBA2_GetPhyStatistics(
       HBA_HANDLE phyHandle,
       SMHBA2_PHYSTATISTICS *pPhyStatistics
);
```

7.5.3.2 Description

The SMHBA2_GetPhyStatistics shall return traffic statistics for a specified phy on a local end port of an HBA.

7.5.3.3 Arguments

Argument **phyHandle** shall be an HBA_HANDLE to a phy.

Argument **pPhyStatistics** shall be a pointer to a structure in which the statistics for the specified phy may be returned.

NOTE 26 - The management client application is assumed to have prior knowledge of the phy type attribute of the phy whose statistics are being retrieved. In addition, the management client application shall also allocate appropriate memory to hold the statistics attributes of the phy prior to invoking SMHBA2_GetPhyStatistics.

7.5.3.4 Return Values

The returned function value shall be as specified in table 40.

Table 40 — Returned Function Values for SMHBA2_GetPhyStatistics

Value	Function result
HBA_STATUS_OK	Phy statistics have been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**phyHandle** is not a phy HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The phy referenced by **phyHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.5.4 SMHBA2_GetFIPStatistics

7.5.4.1 Format

```
HBA_STATUS SMHBA2_GetFIPStatistics{
       HBA_HANDLE FCoECtlrHandle,
       SMHBA2_FIPSTATISTICS *pFIPStatistics
}
```

7.5.4.2 Description

The SMHBA2_GetFIPStatistics function shall return traffic statistics for a specified FCoE Controller.

7.5.4.3 Arguments

Argument **FCoECtlrHandle** shall be an HBA_HANDLE of an FCoE Controller.

Argument **pFIPStatistics** shall be a pointer to a structure in which the statistics for the specified FCoE Controller may be returned.

7.5.4.4 Return Values

The returned function value shall be as specified in table 41.

Table 41 — Returned Function Values for SMHBA2_GetFIPStatistics

Value	Function result
HBA_STATUS_OK	FIP statistics have been returned.
HBA_STATUS_ERROR_INVALID_HANDLE	**FCoECtlrHandle** is not an FCoE Controller HBA_HANDLE.
HBA_STATUS_ERROR_NOT_SUPPORTED	The FCoE Controller referenced by **FCoECtlrHandle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.6 Fabric and Domain Management Functions

7.6.1 HBA_SendCTPassThruV2

7.6.1.1 Format

```
HBA_STATUS HBA_SendCTPassThruV2(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       void *pReqBuffer,
       HBA_UINT32 ReqBufferSize,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize
);
```

7.6.1.2 Description

The HBA_SendCTPassThruV2 function shall send a CT request payload. An HBA should decode this CT_IU request per FC-GS-7, routing the CT frame in a fabric according to the GS_TYPE field within the CT frame.

7.6.1.3 Arguments

Argument **handle** shall be a handle to an HBA through which to send the CT request.

Argument **hbaPortWWN** shall be the N_Port_Name of the local end port through which to send the CT request.

Argument **pReqBuffer** shall be a pointer to a buffer containing the full CT payload, including the CT header, to be sent as defined in FC-GS-7 with the byte ordering as defined in (see FC-LS-2).

Argument **ReqBufferSize** shall be the size of the full CT payload including the CT header, in bytes.

Argument **pRspBuffer** shall be a pointer to a buffer for the CT response.

Argument **RspBufferSize** shall be a pointer to the size of the buffer for the CT response payload in bytes.

7.6.1.4 Return Values

The returned function value shall be as specified in table 42.

Table 42 — Returned Function Values for HBA_SendCTPassThruV2

Value	Function result
HBA_STATUS_OK	The complete reply to the CT Passthrough command has been returned.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA referenced by **handle** does not contain an end port with N_Port_Name **hbaPortWWN**.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function, or the local end port referenced by **hbaportWWN** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

The buffer pointed to by **pRspBuffer** shall contain the CT response payload including the CT header received in response to the frame sent, as defined in FC-GS-7 with the byte ordering as defined in FC-FS-3. If the size of the actual response exceeds the size of the response buffer, trailing data shall be truncated from the response so that the returned data equals the size of the buffer.

The value pointed to by **pRspBufferSize** shall be the size (in bytes) of the actual response data.

7.6.2 HBA_SetRNIDMgmtInfo

7.6.2.1 Format

```
HBA_STATUS HBA_SetRNIDMgmtInfo(
       HBA_HANDLE handle,
       HBA_MGMTINFO info
);
```

7.6.2.2 Description

The HBA_SetRNIDMgmtInfo function shall set the data returned from the HBA in response to an RNID ELS (see FC-LS-2).

7.6.2.3 Arguments

Argument **handle** shall be a handle to an HBA.

Argument **info** shall be a structure containing the information for this HBA to return in a Specific Node Identification Data Format DFh of an RNID Accept (see FC-LS-2).

7.6.2.4 Return Values

The returned function value shall be as specified in table 43.

Table 43 — Returned Function Values for HBA_SetRNIDMgmtInfo

Value	Function result
HBA_STATUS_OK	The RNID reply information has been set as requested.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function, or the local end port referenced by **hbaportWWN** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.6.3 HBA_GetRNIDMgmtInfo

7.6.3.1 Format

```
HBA_STATUS HBA_GetRNIDMgmtInfo(
       HBA_HANDLE handle,
       HBA_MGMTINFO * pInfo
);
```

7.6.3.2 Description

The HBA_GetRNIDMgmtInfo function shall return the RNID (Request Node Identification Information Data) from the HBA (see FC-LS-2).

7.6.3.3 Arguments

Argument **handle** shall be a handle to an HBA.

Argument **pInfo** shall be a pointer to a structure in which to return the information that this HBA returns in a Specific Node Identification Data Format DFh of an RNID Accept (see FC-LS-2).

7.6.3.4 Return Values

The returned function value shall be as specified in table 44.

Table 44 — Returned Function Values for HBA_GetRNIDMgmtInfo

Value	Function result
HBA_STATUS_OK	The RNID reply information has been returned.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function, or the local end port referenced by **hbaportWWN** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

The structure pointed to by **pInfo** shall contain the information that the HBA identified by **handle** returns in a Specific Node Identification Data Format DFh of an RNID Accept (see FC-LS-2).

7.6.4 HBA_SendRNIDV2

7.6.4.1 Format

```
HBA_STATUS HBA_SendRNIDV2(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN destWWN,
       HBA_UINT32 destFCID,
       HBA_UINT32 NodeIdDataFormat,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize
);
```

7.6.4.2 Description

The HBA_SendRNIDV2 function shall send an RNID ELS to another FC_Port requesting a specified Node Identification Data Format.

Parameter **destFCID** may be set to allow the RNID ELS to be sent to an FC_Port that may not be registered with the Name Server. If **destFCID** is set to 000000h, then the parameter shall be ignored. If **destFCID** is not zero, the HBA API library shall use the following rules to verify that the **destWWN**/ **destFCID** pair match in order to limit visibility that may violate scoping mechanisms (e.g., soft zoning):

a) if the **destWWN**/**destFCID** pair matches a discovered port, the RNID shall be sent;
b) if there is no discovered port with the **destWWN** or **destFCID**, then the RNID shall be sent;
c) if there is a discovered port with the **destWWN**, but the **destFCID** does not match, then the request shall be rejected; and
d) on completion of the HBA_SendRNIDV2, if the Common Identification Data Length is nonzero in the RNID response, the API library shall compare the N_Port_Name in the Common Identification Data of the RNID response with **destWWN** and shall fail the operation without returning the response data if they do not match.

7.6.4.3 Arguments

Argument **handle** shall be a handle to an HBA through which the ELS shall be sent.

Argument **hbaPortWWN** shall be the N_Port_Name of the local end port through which the ELS shall be sent.

Argument **destWWN** shall be the N_Port_Name of the remote end port to which the ELS shall be sent.

Argument **destFCID** shall be the address identifier of the destination to which the ELS is sent if **destFCID** is nonzero. Argument **destFCID** shall be ignored if **destFCID** is zero.

Argument **NodeIdDataFormat** shall be a valid value for Node Identification Data Format (see FC-LS-2).

Argument **pRspBuffer** shall be a pointer to a buffer to receive the ELS response.

Argument **pRspBufferSize** shall be a pointer to the size in bytes of **pRspBuffer**.

7.6.4.4 Return Values

The returned function value shall be as specified in table 45.

Table 45 — Returned Function Values for HBA_SendRNIDV2

Value	Function result
HBA_STATUS_OK	The complete LS_ACC to the RNID ELS has been returned.
HBA_STATUS_ERROR_ELS_REJECT	The RNID ELS was rejected by the destination end port.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA referenced by **handle** does not contain an Nx_Port with N_Port_Name **hbaPortWWN**.
HBA_STATUS_ERROR_ILLEGAL_FCID	The **destWWN**/**destFCID** pair conflicts with a discovered N_Port_Name/address identifier pair known by the HBA referenced by **handle**, or the N_Port_Name in the RNID response does not match the **destWWN**.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function, or the local end port referenced by **hbaportWWN** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

The buffer pointed to by **pRspBuffer** shall contain the payload data from the RNID Reply (see FC-LS-2). If the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size pointed to by **pRspBufferSize** at entry to the function, the returned data shall be truncated to the size pointed to by **pRspBufferSize** at entry to the function.

The value pointed to by **pRspBufferSize** shall be the size in bytes of the complete ELS reply payload. This may exceed the size pointed to by **pRspBufferSize** at entry to the function. This shall indicate that the data in **pRspBuffer** has been truncated.

7.6.5 HBA_SendSRL

7.6.5.1 Format

```
HBA_STATUS HBA_SendSRL (
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN wwn,
       HBA_UINT32 domain,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize
);
```

7.6.5.2 Description

The HBA_SendSRL function shall send a Scan Remote Loop (SRL) Extended Link Service through the specified HBA to a specified domain controller (see FC-LS-2).

7.6.5.3 Arguments

Argument **handle** shall be a handle to an HBA through which the ELS shall be sent.

Argument **hbaPortWWN** shall be the N_Port_Name of the local end port through which the ELS shall be sent.

Argument **wwn** shall be the FC_Port Name_Identifier of the FL_Port for the loop to be scanned if **wwn** is nonzero. The ELS shall be sent to the domain controller associated with the named FL_Port. Argument **wwn** shall be ignored if **wwn** is zero.

Argument **domain** shall be a domain number for which all loops shall be scanned if **wwn** is zero. The ELS shall be sent to the domain controller of the domain. Argument **domain** shall be ignored if **wwn** is nonzero.

Argument **pRspBuffer** shall be a pointer to a buffer to receive the ELS response.

Argument **pRspBufferSize** shall be a pointer to the size in bytes of **pRspBuffer**. No valid response exceeds eight bytes.

7.6.5.4 Return Values

The returned function value shall be as specified in table 46.

Table 46 — Returned Function Values for HBA_SendSRL

Value	Function result
HBA_STATUS_OK	The complete LS_ACC to the SRL ELS has been returned.
HBA_STATUS_ERROR_ELS_REJECT	The SRL ELS was rejected by the destination end port.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA referenced by **handle** does not contain an Nx_Port with N_Port_Name **hbaPortWWN**.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function, or the local end port referenced by **hbaportWWN** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

The buffer pointed to by **pRspBuffer** shall contain the payload data from the SRL Reply (see FC-LS-2). If the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size pointed to by **pRspBufferSize** at entry to the function, the returned data shall be truncated to the size pointed to by **pRspBufferSize** at entry to the function.

The value pointed to by **pRspBufferSize** shall be the size in bytes of the complete ELS reply payload. This may exceed the size pointed to by **pRspBufferSize** at entry to the function. This shall indicate that the data in **pRspBuffer** has been truncated.

7.6.6 HBA_SendLIRR

7.6.6.1 Format

```
HBA_STATUS HBA_SendLIRR (
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN destWWN,
       HBA_UINT8 function,
       HBA_UINT8 type,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize
);
```

7.6.6.2 Description

The HBA_SendLIRR function shall send a Link Incident Record Registration (LIRR) Extended Link Service through the specified HBA end port to a specified remote end port (see FC-LS-2). The results of independent use of LIRR by different software controlling the same end port (e.g., two applications

compliant with this standard or an application compliant with this standard and an end port driver) are unpredictable, and may include unexpected link events and changes or losses of link event registration.

NOTE 27 - The unpredictable behavior may be controlled by coordinating the design of the software (e.g., each software module registers for different link event sources and tolerates unexpected link events, or HBA software defers all link event management to an application that calls HBA_SendLIRR).

7.6.6.3 Arguments

Argument **handle** shall be a handle to an HBA through which the ELS shall be sent.

Argument **hbaPortWWN** shall be the N_Port_Name of the local end port through which the ELS shall be sent.

Argument **destWWN** shall be the FC_Port Name_Identifier of the remote FC_Port to which the ELS shall be sent. If this is zero, the destination shall be the Well-known address of the Management Service.

Argument **function** shall be the code for the registration function to be performed. See FC-LS-2 for permitted values and their meanings.

Argument **type** shall be the FC-4 device TYPE (see FC-FS-3) for which specific link incident information requested if **type** is nonzero. Only the common link incident information is requested if **type** is zero.

Argument **pRspBuffer** shall be a pointer to a buffer to receive the ELS response.

Argument **pRspBufferSize** shall be a pointer to the size in bytes of **pRspBuffer**. No valid response exceeds eight bytes.

7.6.6.4 Return Values

The returned function value shall be as specified in table 47.

Table 47 — Returned Function Values for HBA_SendLIRR

Value	Function result
HBA_STATUS_OK	The complete LS_ACC to the LIRR ELS has been returned.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function, or the local end port referenced by **hbaportWWN** does not support this function.
HBA_STATUS_ERROR_ELS_REJECT	The LIRR ELS was rejected by the destination end port.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA referenced by **handle** does not contain an Nx_Port with N_Port_Name **hbaPortWWN**.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

The buffer pointed to by **pRspBuffer** shall contain the payload data from the LIRR Reply (see FC-LS-2). If the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size

pointed to by **pRspBufferSize** at entry to the function, the returned data shall be truncated to the size pointed to by **pRspBufferSize** at entry to the function.

The value pointed to by **pRspBufferSize** shall be the size in bytes of the complete ELS reply payload. This may exceed the size pointed to by **pRspBufferSize** at entry to the function. This shall indicate that the data in **pRspBuffer** has been truncated.

7.6.7 HBA_SendRLS

7.6.7.1 Format

```
HBA_STATUS HBA_SendRLS (
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN destWWN,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize
);
```

7.6.7.2 Description

The HBA_SendRLS function shall send a Read Link Error Status Block (RLS) Extended Link Service through the specified HBA end port to request a specified remote FC_Port to return the Link Error Status Block associated with the destination FC_Port Name_Identifier (see FC-LS-2).

7.6.7.3 Arguments

Argument **handle** shall be a handle to an HBA through which the ELS shall be sent.

Argument **hbaPortWWN** shall be the N_Port_Name of the local end port through which the ELS shall be sent.

Argument **destWWN** shall be the FC_Port Name_Identifier of the remote FC_Port to which the ELS shall be sent.

Argument **pRspBuffer** shall be a pointer to a buffer to receive the ELS response.

Argument **pRspBufferSize** shall be a pointer to the size in bytes of **pRspBuffer**. No valid response exceeds 28 bytes.

7.6.7.4 Return Values

The returned function value shall be as specified in table 48.

Table 48 — Returned Function Values for HBA_SendRLS

Value	Function result
HBA_STATUS_OK	The complete LS_ACC to the RLS ELS has been returned.
HBA_STATUS_ERROR_ELS_REJECT	The RLS ELS was rejected by the destination end port.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA referenced by **handle** does not contain an Nx_Port with N_Port_Name **hbaPortWWN**.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function, or the local end port referenced by **hbaportWWN** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

The buffer pointed to by **pRspBuffer** shall contain the payload data from the RLS Reply (see FC-LS-2). If the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size pointed to by **pRspBufferSize** at entry to the function, the returned data shall be truncated to the size pointed to by **pRspBufferSize** at entry to the function.

The value pointed to by **pRspBufferSize** shall be the size in bytes of the complete ELS reply payload. This may exceed the size pointed to by **pRspBufferSize** at entry to the function. This shall indicate that the data in **pRspBuffer** has been truncated.

7.6.8 SMHBA_SendTEST

7.6.8.1 Format

```
HBA_STATUS SMHBA_SendTEST(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN destWWN,
       HBA_UINT32 destFCID,
       void *pReqBuffer,
       HBA_UINT32 ReqBufferSize
);
```

7.6.8.2 Description

The SMHBA_SendTEST function shall send a TEST ELS with a specified Payload to another FC_port. There is no response from a TEST ELS.

Parameter **destFCID** may be set to allow the TEST ELS to be sent to an FC_Port that may not be registered with the Name Server. If **destFCID** is set to 000000h, then the parameter is ignored. If **destFCID** is not zero, the HBA API library shall verify that the **destWWN/destFCID** pair match in order to limit visibility that may violate scoping mechanisms (e.g., soft zoning):

a) If the **destWWN/destFCID** pair matches a discovered port, the TEST shall be sent;
b) If there is no discovered port with the **destWWN** or **destFCID**, then the TEST shall be sent; and
c) If there is a discovered port with the **destWWN**, but the **destFCID** does not match, then the request shall be rejected.

7.6.8.3 Arguments

Argument **handle** shall be a handle to an HBA through which the TEST ELS shall be sent.

Argument **hbaPortWWN** shall be the N_Port_Name of the local end port through which the TEST ELS shall be sent.

Argument **destWWN** shall be the N_Port_Name of the remote end port to which the TEST ELS shall be sent.

Argument **destFCID** shall be the address identifier of the destination to which the ELS is sent if **destFCID** is nonzero. Argument **destFCID** shall be ignored if **destFCID** is zero.

Argument **pReqBuffer** shall be a pointer to a buffer containing the full TEST data, as defined in FC-LS-2.

Argument **pReqBufferSize** shall be the size in bytes of **pReqBuffer**.

7.6.8.4 Return Values

The returned function value shall be as specified in table 49.

Table 49 — Returned Function Values for SMHBA_SendTEST

Value	Function result
HBA_STATUS_OK	The TEST ELS has been transmitted.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA referenced by **handle** does not contain an Nx_Port with N_Port_Name **hbaPortWWN**.
HBA_STATUS_ERROR_ILLEGAL_FCID	The **destWWN**/**destFCID** pair conflicts with a discovered N_Port_Name/address identifier pair known by the HBA referenced by **handle**.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function, or the local end port referenced by **hbaportWWN** does not support this function.
any value in 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.6.9 SMHBA_SendECHO

7.6.9.1 Format

```
HBA_STATUS SMHBA_SendECHO(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN destWWN,
       HBA_UINT32 destFCID,
       void *pReqBuffer,
       HBA_UINT32 ReqBufferSize,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize
);
```

7.6.9.2 Description

The SMHBA_SendECHO function shall send a ECHO ELS with a specified Payload to another FC_port.

Parameter **destFCID** may be set to allow the ECHO ELS to be sent to an FC_Port that may not be registered with the Name Server. If **destFCID** is set to 000000h, then the parameter is ignored. If **destFCID** is not zero, the HBA API library shall verify that the **destWWN/destFCID** pair match in order to limit visibility that may violate scoping mechanisms (e.g., soft zoning):

a) If the **destWWN/destFCID** pair matches a discovered port, the ECHO shall be sent;
b) If there is no discovered port with the **destWWN** or **destFCID**, then the ECHO shall be sent; and
c) If there is a discovered port with the **destWWN**, but the **destFCID** does not match, then the request shall be rejected.

7.6.9.3 Arguments

Argument **handle** shall be a handle to an HBA through which the ECHO ELS shall be sent.

Argument **hbaPortWWN** shall be the N_Port_Name of the local end port through which the ECHO ELS shall be sent.

Argument **destWWN** shall be the N_Port_Name of the remote end port to which the ECHO ELS shall be sent.

Argument **destFCID** shall be the address identifier of the destination to which the ELS is sent if **destFCID** is nonzero. Argument **destFCID** shall be ignored if **destFCID** is zero.

Argument **pReqBuffer** shall be a pointer to a buffer containing the full ECHO data, as defined in FC-LS-2.

Argument **pReqBufferSize** shall be the size in bytes of **pReqBuffer**.

Argument **pRspBuffer** shall be a pointer to a buffer to receive the ELS response.

Argument **pRspBufferSize** shall be a pointer to the size in bytes of **pRspBuffer**.

7.6.9.4 Return Values

The returned function value shall be as specified in table 50 .

Table 50 — Returned Function Values for SMHBA_SendEcho

Value	Function result
HBA_STATUS_OK	The complete LS_ACC to the ECHO ELS has been returned.
HBA_STATUS_ERROR_ELS_REJECT	The ECHO ELS was rejected by the destination end port.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA referenced by **handle** does not contain an Nx_Port with N_Port_Name **hbaPortWWN**.
HBA_STATUS_ERROR_ILLEGAL_FCID	The **destWWN**/**destFCID** pair conflicts with a discovered N_Port_Name/address identifier pair known by the HBA referenced by **handle**.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function, or the local end port referenced by **hbaportWWN** does not support this function.
any value in 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

The buffer pointed to by pRspBuffer shall contain the ECHO Reply (see FC-LS-2). If the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size pointed to by **pRspBufferSize** at entry to the function, the returned data shall be truncated to the size pointed to by **pRspBufferSize** at entry to the function.

The value pointed to by **pRspBufferSize** shall be the size in bytes of the complete ELS reply payload. This may exceed the size pointed to by **pRspBufferSize** at entry to the function. This shall indicate that the data in **pRspBuffer** has been truncated.

7.6.10 SMHBA_SendSMPPassThru

7.6.10.1 Format

```
HBA_UINT32 SMHBA_SendSMPPassThru(
       HBA_HANDLE    handle,
       HBA_WWN       hbaportWWN,
       HBA_WWN       destportWWN,
       HBA_WWN       domainPortWWN,
       void          *pReqBuffer,
       HBA_UINT32    ReqBufferSize,
       void          *pRspBuffer,
       HBA_UINT32    *pRspBufferSize
);
```

7.6.10.2 Description

The SMHBA_SendSMPPassThru function shall send an SMP Request frame with the specified payload through the specified HBA end port to the specified destination port. The function shall return an error if the HBA end port or the destination port is not an SMP Port (see SPL).

Argument **handle** shall be a handle to an HBA through which the SMP Request frame shall be sent.

Argument **hbaportWWN** shall be the Port_Identifier of the local SAS Port through which the SMP Request frame shall be sent.

Argument **destportWWN** shall be the Port_Identifier of the destination SAS Port to which the SMP Request frame shall be sent.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP port discovered through the specified local port on the specified HBA or zero. If domainPortWWN is not zero, the port to query shall be attached to the same SAS domain as the specified SMP port. Argument domainPortWWN shall be ignored if the specified port is not a SAS Port.

Argument **pReqBuffer** shall be the pointer to a buffer containing the SMP Request frame as defined in the SAS Specification (see SPL).

Argument **ReqBufferSize** shall be the size of the SMP Request frame payload in bytes.

Argument **pRspBuffer** shall be the pointer to a buffer for the SMP Response frame (see SPL).

Argument **pRspBufferSize** shall be the pointer to the size of the buffer for the SMP Response payload in bytes.

7.6.10.3 Return Values

The returned function value shall be as specified in table 51.

Table 51 — Returned Function Values for SMHBA_SendSMPPassThru (part 1 of 2)

Value	Function Result
HBA_STATUS_OK	The complete SMP Response to the SMP Request has been returned

Table 51 — Returned Function Values for SMHBA_SendSMPPassThru (part 2 of 2)

Value	Function Result
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support this function, or the local end port referenced by **hbaportWWN** does not support this function, or the destination port referenced by **desportWWN** does not support this function.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA referenced by the **handle** does not contain the port referenced by **hbaportWWN**, or the HBA referenced by the handle and the port referenced by **hbaportWWN** does not contain a destination port referenced by **destportWWN**, or the HBA identified by handle is not able to access an expander SMP target port identified by **domainPortWWN**.
any other	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

The buffer pointed to by **pRspBuffer** shall contain the payload data of the SMP Response frame (see SPL). If the size of the response payload exceeds the size pointed to by **pRspBufferSize** at entry to the function, the returned data shall be truncated to the size pointed to by **pRspBufferSize** at entry to the function.

The value pointed to by **pRspBufferSize** shall be the size in bytes of the complete payload data from the SMP Response frame (see SPL). This may exceed the size pointed to by **pRspBufferSize** at entry to the function. This shall indicate that the data in **pRspBuffer** has been truncated.

7.7 SM-HBA Target Information Functions

7.7.1 SMHBA_GetBindingCapability

7.7.1.1 Format

```
HBA_STATUS SMHBA_GetBindingCapability(
       HBA_HANDLE    handle,
       HBA_WWN       hbaPortWWN,
       HBA_WWN       domainPortWWN,
       SMHBA_BIND_CAPABILITY *pFlags
);
```

7.7.1.2 Description

The SMHBA_GetBindingCapability function shall return the binding capabilities implemented for a specified HBA end port.

7.7.1.3 Arguments

Argument **handle** shall be a handle to an HBA containing the local end port for which to return implemented persistent binding capabilities.

Argument **hbaportWWN** shall be the Name_Identifier of the local FC_Port, or the Port_Identifier of the local SAS Port for which to return implemented persistent binding capabilities.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP port discovered through the specified local port on the specified HBA or zero. If domainPortWWN is not zero, the port to query shall be attached to the same SAS domain as the specified SMP port. Argument domainPortWWN shall be ignored if the specified port is not a SAS Port.

Argument **pFlags** shall point to an SMHBA_BIND_CAPABILITY structure in which to return implemented persistent binding capabilities.

7.7.1.4 Return Values

The returned function value shall be as specified in table 52.

Table 52 — Returned Function Values for SMHBA_GetBindingCapability

Value	Function result
HBA_STATUS_OK	Implemented persistent binding capabilities have been returned.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local end port or discovered FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local end port or discovered SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support persistent binding.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the implemented persistent binding capabilities for the local end port identified by **hbaPortWWN** and **handle** shall be in the structure pointed to by **pFlags**.

7.7.2 SMHBA_GetBindingSupport

7.7.2.1 Format

```
HBA_STATUS SMHBA_GetBindingSupport(
       HBA_HANDLE                  handle,
       HBA_WWN                     hbaPortWWN,
       HBA_WWN                     domainPortWWN,
       SMHBA_BIND_CAPABILITY       *pFlags
);
```

7.7.2.2 Description

The SMHBA_GetBindingSupport function shall return the binding capabilities currently enabled for a specified HBA end port.

7.7.2.3 Arguments

Argument **handle** shall be a handle to an HBA containing the end port for which to return currently enabled persistent binding capabilities.

Argument **hbaportWWN** shall be the Name_Identifier of the local FC_Port, or the Port_Identifier of the local SAS Port for which to return the currently enabled binding capabilities.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP port discovered through the specified local port on the specified HBA or zero. If domainPortWWN is not zero, the port to query shall be attached to the same SAS domain as the specified SMP port. Argument domainPortWWN shall be ignored if the specified port is not a SAS Port.

Argument **pFlags** shall point to an SMHBA_BIND_CAPABILITY structure in which to return currently enabled persistent binding capabilities.

7.7.2.4 Return Values

The returned function value shall be as specified in table 53.

Table 53 — Returned Function Values for SMHBA_GetBindingSupport

Value	Function result
HBA_STATUS_OK	Enabled persistent binding capabilities have been returned.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local end port or discovered FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local end port or discovered SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support persistent binding.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the currently enabled persistent binding capabilities for the local end port identified by **hbaPortWWN** and **handle** shall be in the structure pointed to by **pFlags**.

7.7.3 SMHBA_SetBindingSupport

7.7.3.1 Format

```
HBA_STATUS SMHBA_SetBindingSupport(
       HBA_HANDLE                  handle,
       HBA_WWN                     hbaPortWWN,
       HBA_WWN                     domainPortWWN,
       SMHBA_BIND_CAPABILITY       flags
);
```

7.7.3.2 Description

The SMHBA_SetBindingSupport function shall set the binding capabilities currently enabled for a specified HBA end port to a subset of those that the HBA end port has implemented.

Disabling SMHBA_CAN_BIND_AUTOMAP shall limit the OS visibility of the SAN to those resources explicitly identified in persistent bindings. This standard does not propose any utility to disable other capabilities.

7.7.3.3 Arguments

Argument **handle** shall be a handle to an HBA containing the end port for which to set the currently enabled persistent binding capabilities.

Argument **hbaportWWN** shall be the Name_Identifier of the local FC_Port, or, the Port_Identifier of the local SAS Port for which to set the currently enabled binding capabilities.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP port discovered through the specified local port on the specified HBA or zero. If domainPortWWN is not zero, the port to configure shall be attached to the same SAS domain as the specified SMP port. Argument domainPortWWN shall be ignored if the specified port is not a SAS Port.

Argument **flags** shall be an SMHBA_BIND_CAPABILITY structure indicating persistent binding capabilities to enable.

7.7.3.4 Return Values

The returned function value shall be as specified in table 54.

Table 54 — Returned Function Values for SMHBA_SetBindingSupport (part 1 of 2)

Value	Function result
HBA_STATUS_OK	Persistent binding capabilities identified by **flags** have been enabled.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local end port or discovered FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local end port or discovered SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.

Table 54 — Returned Function Values for SMHBA_SetBindingSupport (part 2 of 2)

Value	Function result
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA end port referenced by **hbaPortWWN** and **handle** does not support persistent binding.
HBA_ERROR_INCAPABLE	The **flags** include a flag for a capability not implemented for the HBA end port referenced by **hbaPortWWN** and **handle**.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.7.4 SMHBA_GetTargetMapping

7.7.4.1 Format

```
HBA_STATUS SMHBA_GetTargetMapping(
       HBA_HANDLE           handle,
       HBA_WWN              hbaPortWWN,
       HBA_WWN              domainPortWWN
       SMHBA_TARGETMAPPING  *pMapping
);
```

7.7.4.2 Description

The SMHBA_GetTargetMapping function shall return the mapping that exists at the time the function call is processed between OS identification of SCSI target ports or logical units and:

a) FCP identification of SCSI target ports;
b) SSP identification of SCSI target ports; or
c) logical units offered by the specified HBA end port.

Space in the pMapping structure permitting, one mapping entry shall be returned for each logical unit represented in the OS by the HBA and one mapping entry shall be returned for each FCP-4 target port or, SSP target port that is represented in the OS but for which no logical units are represented in the OS. No target mappings shall be returned to represent FCP objects or SSP objects that are not represented in the OS (i.e., are unmapped).

The mappings returned shall include a Logical Unit Unique Device Identifier (LUID) for each logical unit that provides one (see SAM-5, FCP-4, SPL, and relevant OS documentation). If the Vital Products Data Device Identification Page (VPD Page 83h, see SPC-4) information for a logical unit provides more than one LUID, the one returned shall be:

1) the type 3 (FC Name_Identifier or Port_Identifier) LUID with the smallest identifier value if any LUID of type 3 is provided;
2) the type 2 (IEEE EUI-64) LUID with the smallest identifier value if any LUID of type 2 is provided and no LUID of type 3 is provided;

3) the type 1 (T10 vendor identification) LUID with the smallest identifier value if any LUID of type 1 is provided and no LUID of type 2 or type 3 is provided;
4) the type 0 (vendor specific) LUID with the smallest identifier value if any LUID of type 0 is provided and no LUID of type 1 or type 2 or type 3 is provided; or
5) any value of which the high order four bytes are all zero if the logical unit provides no LUID.

7.7.4.3 Arguments

Argument **handle** shall be a handle to an HBA containing the end port for which target mappings are requested.

Argument **hbaportWWN** shall be the Name_Identifier of the local FC_Port, or the Port_Identifier of the local SAS Port for which target mappings are requested.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP port discovered through the specified local port on the specified HBA or zero. If domainPortWWN is not zero, the port to query shall be attached to the same SAS domain as the specified SMP port. Argument domainPortWWN shall be ignored if the specified port is not a SAS Port.

Argument **pMapping** shall be a pointer to an SMHBA_TARGETMAPPING structure. The size of this structure shall be limited by the NumberOfEntries value within the structure.

7.7.4.4 Return Values

The returned function value shall be as specified in table 55.

Table 55 — Returned Function Values for SMHBA_GetTargetMapping (part 1 of 2)

Value	Function result
HBA_STATUS_OK	All mapping entries have been returned.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some mapping entries may have been returned.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local end port or discovered FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local end port or discovered SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.

Table 55 — Returned Function Values for SMHBA_GetTargetMapping (part 2 of 2)

Value	Function result
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support target mapping.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK or HBA_STATUS_ERROR_MORE_DATA, the structure pointed to by **pMapping** shall contain mapping information from OS identifications of SCSI logical units to FCP identifications or, SSP identifications, of logical units for the local end port identified by **hbaPortWWN** and **handle** (see SAM-5, FCP-4, SPL, and relevant OS documentation). The number of entries in the structure shall be the minimum of the number of entries specified at function call or the full mapping. The value of the NumberOfEntries field of the returned **pMapping** structure shall be the total number of mappings the end port has established even when the function returns an error because the buffer is too small to return all of the established mappings. An application may either allocate a sufficiently large buffer and check this value after a read, or do a read of the NumberOfEntries value separately and allocate a new buffer given the size to accommodate the entire mapping structure.

7.7.5 SMHBA_GetPersistentBinding

7.7.5.1 Format

```
HBA_STATUS SMHBA_GetPersistentBinding(
       HBA_HANDLE    handle,
       HBA_WWN       hbaPortWWN,
       HBA_WWN       domainPortWWN,
       SMHBA_BINDING *binding
);
```

7.7.5.2 Description

The SMHBA_GetPersistentBinding function shall return persistent bindings for the specified HBA end port between SCSI IDs and:

a) FCP-4 target ports;
b) SSP target ports; or
c) SCSI logical units.

The binding information may include bindings to LUIDs.

7.7.5.3 Arguments

Argument **handle** shall be a handle to an HBA containing the end port for which to return persistent binding.

Argument **hbaportWWN** shall be the Name_Identifier of the local FC_Port or, the Port_Identifier of the local SAS Port for which to return persistent binding.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP port discovered through the specified local port on the specified HBA or zero. If domainPortWWN is not zero, the port to query shall be

attached to the same SAS domain as the specified SMP port. Argument domainPortWWN shall be ignored if the specified port is not a SAS Port.

Argument **binding** shall be a pointer to an SMHBA_BINDING structure. The NumberOfEntries field in the structure shall limit the number of entries that shall be returned.

7.7.5.4 Return Values

The returned function value shall be as specified in table 56.

Table 56 — Returned Function Values for SMHBA_GetPersistentBinding

Value	Function result
HBA_STATUS_OK	All binding entries have been returned.
HBA_STATUS_ERROR_MORE_DATA	More space in the buffer is required to contain mapping information. Some binding entries may have been returned.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local end port or discovered FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local end port or discovered SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support persistent binding.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK or HBA_STATUS_ERROR_MORE_DATA, the structure pointed to by **binding** shall contain binding information from OS identifications of SCSI logical units to FCP or SSP and LUID identifications of logical units for the local end port identified by **hbaPortWWN** and **handle** (see SAM-5, FCP-4, SPL, and relevant OS documentation). The number of entries in the returned **binding** structure shall be the minimum of the number of entries specified at function call or the full set of bindings. The NumberOfEntries field shall contain the total number of bindings established by the end port. An application may either call SMHBA_GetPersistentBinding with NumberOfEntries set to zero to retrieve the number of entries available, or allocate a sufficiently large buffer to retrieve entries at first call. The Status field of each SMHBA_BINDINGENTRY substructure shall be zero.

7.7.6 SMHBA_SetPersistentBinding

7.7.6.1 Format

```
HBA_STATUS SMHBA_SetPersistentBinding(
       HBA_HANDLE           handle,
       HBA_WWN              hbaPortWWN,
       HBA_WWN              domainPortWWN,
       const SMHBA_BINDING  *binding
);
```

7.7.6.2 Description

The SMHBA_SetPersistentBinding function shall set additional persistent bindings for the specified HBA end port between SCSI IDs and:

a) FCP-4 target ports;
b) SSP target ports; or
c) SCSI logical units.

It shall accept extended bindings to LUIDs. Bindings already in effect shall remain in effect. A request for a binding to the same local OS SCSI ID as a binding that is already in effect shall cause the returned function value to be an error code (see 7.7.6.4). Each requested binding may succeed or fail independently of the others.

Persistent bindings established by this call shall not cause change of a target mapping until reinitialization of the OS, HBA, and/or fabric. The effects on target mappings of establishing persistent bindings by other means (e.g., vendor specific API or management utility) is not specified by this standard.

7.7.6.3 Arguments

Argument **handle** shall be a handle to an HBA containing the end port for which to set persistent binding.

Argument **hbaportWWN** shall be the Name_Identifier of the local FC_Port, or the Port_Identifier of the local SAS Port for which to set persistent binding.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP port discovered through the specified local port on the specified HBA or zero. If domainPortWWN is not zero, the port to configure shall be attached to the same SAS domain as the specified SMP port. Argument domainPortWWN shall be ignored if the specified port is not a SAS Port.

Argument **binding** shall be a pointer to an SMHBA_BINDING structure. The NumberOfEntries field in the structure shall determine the number of requested entries in the structure.

7.7.6.4 Return Values

The returned function value shall be as specified in table 57.

Table 57 — Returned Function Values for SMHBA_SetPersistentBinding

Value	Function result
HBA_STATUS_OK	At least one persistent binding has been set.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local end port or discovered FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local end port or discovered SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support persistent binding.
HBA_STATUS_ERROR_LOCAL_SCSIID_BOUND	A persistent binding set request included a local SCSI ID that was already bound.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

The structure pointed to by **binding** shall indicate the success or failure of setting the binding requested by each SMHBA_BINDINGENTRY substructure by setting the value of the Status field in the substructure to a value defined in 6.2.

7.7.7 SMHBA_RemovePersistentBinding

7.7.7.1 Format

```
HBA_STATUS SMHBA_RemovePersistentBinding(
       HBA_HANDLE           handle,
       HBA_WWN              hbaPortWWN,
       HBA_WWN              domainPortWWN,
       const SMHBA_BINDING  *binding
);
```

7.7.7.2 Description

The SMHBA_RemovePersistentBinding function shall remove one or more persistent bindings to specified SCSI IDs for the specified HBA end port. A persistent binding shall be removed if and only if both the local SCSI ID and SMHBA_SCSIENTRY match a binding specified in the arguments. The removal of any binding shall not affect other persistent bindings.

Persistent bindings removed by this call shall not cause change of a target mapping until reinitialization of the OS, HBA, and/or fabric. The effects on target mappings of removing persistent bindings by other means (e.g., vendor specific API or management utility) is not specified by this standard.

7.7.7.3 Arguments

Argument **handle** shall be a handle to an HBA containing the end port from which to remove persistent bindings.

Argument **hbaportWWN** shall be the Name_Identifier of the local FC_Port, or the Port_Identifier of the local SAS Port for which to remove persistent bindings.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP port discovered through the specified local port on the specified HBA or zero. If domainPortWWN is not zero, the port to configure shall be attached to the same SAS domain as the specified SMP port. Argument domainPortWWN shall be ignored if the specified port is not a SAS Port.

Argument **binding** shall be a pointer to a SMHBA_BINDING structure indicating the bindings for which removal is requested. The NumberOfEntries field in the structure shall determine the number of requested entries in the structure.

7.7.7.4 Return Values

The returned function value shall be as specified in table 58.

Table 58 — Returned Function Values for SMHBA_RemovePersistentBinding

Value	Function result
HBA_STATUS_OK	At least one persistent binding has been removed.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local end port or discovered FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local end port or discovered SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support persistent binding.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

The structure pointed to by **binding** shall indicate the success or failure of removing the binding requested by each SMHBA_BINDINGENTRY substructure by setting the value of the Status field in the substructure to a value defined in 6.2.

7.7.8 SMHBA_RemoveAllPersistentBindings

7.7.8.1 Format

```
HBA_STATUS SMHBA_RemoveAllPersistentBindings(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN domainPortWWN
);
```

7.7.8.2 Description

The SMHBA_RemoveAllPersistentBindings function shall remove all persistent bindings for a specified HBA end port.

Persistent bindings removed by this call shall not cause change of a target mapping until reinitialization of the OS, HBA, and/or fabric. The effects on target mappings of removing persistent bindings by other means (e.g., vendor specific API or management utility) is not specified.

7.7.8.3 Arguments

Argument **handle** shall be a handle to an HBA containing the end port from which to remove all persistent bindings.

Argument **hbaportWWN** shall be the Name_Identifier of the local FC_Port, or the Port_Identifier of the local SAS Port for which to remove all persistent bindings.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP port discovered through the specified local port on the specified HBA or zero. If domainPortWWN is not zero, the port to configure shall be attached to the same SAS domain as the specified SMP port. Argument domainPortWWN shall be ignored if the specified port is not a SAS Port.

7.7.8.4 Return Values

The returned function value shall be as specified in table 59.

Table 59 — Returned Function Values for SMHBA_RemoveAllPersistentBindings

Value	Function result
HBA_STATUS_OK	All persistent bindings have been removed.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local end port or discovered FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local end port or discovered SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA referenced by **handle** does not support persistent binding.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

7.7.9 SMHBA_GetLUNStatistics

7.7.9.1 Format

```
HBA_STATUS SMHBA_GetLUNStatistics(
       HBA_HANDLE handle,
       const HBA_SCSIID *lunit,
       SMHBA_PROTOCOLSTATISTICS *statistics
);
```

7.7.9.2 Description

The SMHBA_GetLUNStatistics function shall return traffic statistics for a specific OS SCSI logical unit provided via the FCP protocol or, SSP Protocol on a specific local HBA.

7.7.9.3 Arguments

Argument **handle** shall be a handle to an HBA for which to return FCP-4 statistics or SSP statistics.

Argument **lunit** shall be a pointer to a structure specifying the OS SCSI logical unit for which FCP-4 statistics or, SSP statistics are requested.

Argument **statistics** shall be a pointer to a SMHBA_ProtocolStatistics structure in which the statistics for the specified logical unit may be returned.

7.7.9.4 Return Values

The returned function value shall be as specified in table 60.

Table 60 — Returned Function Values for SMHBA_GetLUNStatistics

Value	Function result
HBA_STATUS_OK	Protocol (FCP-4 or SSP) statistics have been returned.
HBA_STATUS_ERROR_INVALID_LUN	The HBA identified by **handle** does not support the SCSI logical unit identified by **lunit**.
HBA_STATUS_ERROR_NOT_A_SCSIDEVICE	The HBA identified by **handle** does not support SCSI.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the structure pointed to by **statistics** shall contain the FCP-4 statistics, or SSP Statistics for the HBA identified by **handle** accessing the SCSI logical unit identified by **lunit**.

7.8 SCSI Information Functions

7.8.1 SMHBA_ScsiInquiry

7.8.1.1 Format

```
HBA_STATUS SMHBA_ScsiInquiry (
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN discoveredPortWWN,
       HBA_WWN domainPortWWN;
       SMHBA_SCSILUN smhbaLUN,
       HBA_UINT8 CDB_Byte1,
       HBA_UINT8 CDB_Byte2,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize,
       HBA_UINT8 *pScsiStatus,
       void *pSenseBuffer,
       HBA_UINT32 *pSenseBufferSize
);
```

7.8.1.2 Description

The SMHBA_ScsiInquiry function shall send a SCSI INQUIRY command (see SPC-4) to a SCSI logical unit through a remote FCP_Port or SSP_Port.

A SCSI command shall not be sent to a discovered FCP_Port or SSP_Port if that does not have SCSI target port functionality. An HBA API library shall not cause a SCSI overlapped command condition (see SAM-5). Proper use of tagged commands (see SAM-5) is an acceptable means of avoiding a SCSI overlapped command condition with SCSI logical units that support tagged commands.

7.8.1.3 Arguments

Argument **handle** shall be a handle to an HBA through which the SCSI INQUIRY command shall be sent.

Argument **hbaPortWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port on the specified HBA through which the SCSI INQUIRY command shall be sent.

Argument **discoveredPortWWN** shall be the Name_Identifier of the FCP_Port, or the Port_Identifier of the SSP_Port to which the SCSI INQUIRY command shall be sent.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP target port discovered through the specified local port on the specified HBA. It shall have a value of 0 if no expander SMP target ports were discovered. It shall be ignored if the local port is not a SAS Port.

Argument **smhbaLUN** shall be the SCSI LUN to which the SCSI INQUIRY command shall be sent.

Argument **CDB_Byte1** shall be the second byte of the CDB for the SCSI INQUIRY command. This contains control flag bits. At the time this standard was written, the effects of the value of CDB_Byte1 on a SCSI INQUIRY command were as shown in table 61.

Table 61 — Values for CDB_Byte1

CDB_Byte1 value	Effect
0	Request the standard SCSI INQUIRY data
1	Request the vital product data (EVPD) specified by CDB_Byte2
2	Request command support data (CmdDt) for the command specified in CDB_Byte2
other values	May cause SCSI CHECK CONDITION

Argument **CDB_Byte2** shall be the third byte of the CDB for the SCSI INQUIRY command. If **CDB_Byte1** is set to one, **CDB_Byte2** shall be the Vital Product Data page code to request. If **CDB_Byte1** is set to two, **CDB_Byte2** shall be the Operation Code of the command support data requested. For other values of **CDB_Byte1**, the value of **CDB_Byte2** is unspecified, and values other than zero may cause a SCSI Check Condition.

Argument **pRspBuffer** shall be a pointer to a buffer to receive the SCSI INQUIRY command response.

Argument **pRspBufferSize** shall be a pointer to the size in bytes of the buffer to receive the SCSI INQUIRY command response.

Argument **pScsiStatus** shall be a pointer to a buffer to receive SCSI status.

Argument **pSenseBuffer** shall be a pointer to a buffer to receive SCSI sense data.

Argument **pSenseBufferSize** shall be a pointer to the size in bytes of the buffer to receive SCSI sense data.

7.8.1.4 Return Values

The returned function value shall be as specified in table 62.

Table 62 — Returned Function Values for SMHBA_ScsiInquiry

Value	Function result
HBA_STATUS_OK	The complete payload of a reply to the SCSI INQUIRY command has been returned.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local FC_Port with Name_Identifier identified by **hbaPortWWN**, or the HBA identified by **handle** is not able to access a local SAS Port with Port_Identifier identified by **hbaPortWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_NOT_A_TARGET	The remote end port identified by **discoveredPortWWN** does not have SCSI target port functionality.
HBA_STATUS_ERROR_TARGET_BUSY	Unable to send the requested command without causing a SCSI overlapped command condition.
HBA_STATUS_SCSI_CHECK_CONDITION	Returned SCSI status indicates a SCSI CHECK CONDITION.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the function value is HBA_STATUS_OK, the buffer pointed to by **pRspBuffer** shall contain the response to the SCSI INQUIRY command.

The value pointed to by **pRspBufferSize** shall be the size in bytes of the response returned by the command. This shall not exceed the size passed as an argument at this pointer.

The value pointed to by **pScsiStatus** shall be the SCSI status (see SAM-5). If the function value is HBA_STATUS_OK or HBA_STATUS_SCSI_CHECK_CONDITION, the value of the SCSI status may be interpreted based on the SAM-4 standard. A SCSI status of GOOD indicates a SCSI response is in the buffer pointed to by **pRspBuffer**. A SCSI status of CHECK CONDITION indicates no value is stored in the buffer pointed to by **pRspBuffer**, and the buffer pointed to by **pSenseBuffer** shall contain failure information if available. All other SCSI status codes should be interpreted by reference to SAM-5.

If the function value is HBA_STATUS_SCSI_CHECK_CONDITION, the buffer pointed to by **pSenseBuffer** shall contain the sense data for the command.

The value pointed to by **pSenseBufferSize** shall be the size in bytes of the sense information returned by the command. This shall not exceed the size passed as an argument at this pointer.

7.8.2 SMHBA_ScsiReportLuns

7.8.2.1 Format

```
HBA_STATUS SMHBA_ScsiReportLuns(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN discoveredPortWWN,
       HBA_WWN domainPortWWN;
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize,
       HBA_UINT8 *pScsiStatus,
       void *pSenseBuffer,
       HBA_UINT32 *pSenseBufferSize
);
```

7.8.2.2 Description

The SMHBA_ScsiReportLuns function shall send a SCSI REPORT LUNS command (see SPC-4) to SCSI logical unit zero through a remote FCP_Port or SSP_Port.

A SCSI command shall not be sent through a remote FCP_Port or SSP_Port that does not have SCSI target port functionality. An HBA API library shall not cause a SCSI overlapped command condition (see SAM-5). Proper use of tagged commands (see SAM-5) is an acceptable means of avoiding a SCSI overlapped command condition with SCSI logical units that support tagged commands.

7.8.2.3 Arguments

Argument **handle** shall be a handle to an HBA through which the SCSI REPORT LUNS command shall be sent.

Argument **hbaPortWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port on the specified HBA through which the SCSI REPORT LUNS command shall be sent.

Argument **discoveredPortWWN** shall be the Name_Identifier of the FCP_Port, or the Port_Identifier of the SSP_Port to which the SCSI REPORT LUNS command shall be sent.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP target port discovered through the specified local port on the specified HBA. It shall have a value of 0 if no expander SMP target ports were discovered. It shall be ignored if the local port is not a SAS Port.

Argument **pRspBuffer** shall be a pointer to a buffer to receive the SCSI REPORT LUNS command response.

Argument **pRspBufferSize** shall be a pointer to the size in bytes of the buffer to receive the SCSI REPORT LUNS command response.

Argument **pScsiStatus** shall be a pointer to a buffer to receive SCSI status.

Argument **pSenseBuffer** shall be a pointer to a buffer to receive SCSI sense data.

Argument **pSenseBufferSize** shall be a pointer to the size in bytes of the buffer to receive SCSI sense data.

7.8.2.4 Return Values

The returned function value shall be as specified in table 63.

Table 63 — Returned Function Values for SMHBA_ScsiReportLuns

Value	Function result
HBA_STATUS_OK	The complete payload of a reply to the SCSI REPORT LUNS command has been returned.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_NOT_A_TARGET	The remote end port identified by **discoveredPortWWN** does not have SCSI target port functionality.
HBA_STATUS_ERROR_TARGET_BUSY	Unable to send the requested command without causing a SCSI overlapped command condition.
HBA_STATUS_SCSI_CHECK_CONDITION	Returned SCSI status indicates a SCSI CHECK CONDITION.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the function value is HBA_STATUS_OK, the buffer pointed to by **pRspBuffer** shall contain the response to the SCSI REPORT LUNS command.

The value pointed to by **pRspBufferSize** shall be the size in bytes of the response returned by the command. This shall not exceed the size passed as an argument at this pointer.

The value pointed to by **pScsiStatus** shall be the SCSI status (see SAM-5). If the function value is HBA_STATUS_OK or HBA_STATUS_SCSI_CHECK_CONDITION, the value of the SCSI status may be interpreted based on the SAM-4 standard. A SCSI status of GOOD indicates a SCSI response is in the buffer pointed to by **pRspBuffer**. A SCSI status of CHECK CONDITION indicates no value is stored in the buffer pointed to by **pRspBuffer**, and the buffer pointed to by **pSenseBuffer** shall contain failure information if available. All other SCSI status codes should be interpreted by reference to SAM-5.

If the function value is HBA_STATUS_SCSI_CHECK_CONDITION, the buffer pointed to by **pSenseBuffer** shall contain the sense data for the command.

The value pointed to by **pSenseBufferSize** shall be the size in bytes of the sense information returned by the command. This shall not exceed the size passed as an argument at this pointer.

7.8.3 SMHBA_ScsiReadCapacity

7.8.3.1 Format

```
HBA_STATUS SMHBA_ScsiReadCapacity(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN discoveredPortWWN,
       HBA_WWN domainPortWWN,
       SMHBA_SCSILUN smhbaLUN,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize,
       HBA_UINT8 *pScsiStatus,
       void *pSenseBuffer,
       HBA_UINT32 *pSenseBufferSize
);
```

7.8.3.2 Description

The SMHBA_ScsiReadCapacity function shall send a SCSI READ CAPACITY command (see SBC-3) to a SCSI logical unit through a remote FCP_Port or SSP_Port.

A SCSI command shall not be sent through a remote FCP_Port or SSP_Port that does not have SCSI target port functionality. An HBA API library shall not cause a SCSI overlapped command condition (see SAM-5). Proper use of tagged commands (see SAM-5) is an acceptable means of avoiding a SCSI overlapped command condition with SCSI logical units that support tagged commands.

7.8.3.3 Arguments

Argument **handle** shall be a handle to an HBA through which the SCSI READ CAPACITY command shall be sent.

Argument **hbaPortWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port on the specified HBA through which the SCSI READ CAPACITY command shall be sent.

Argument **discoveredPortWWN** shall be the Name_Identifier of the FCP_Port, or the Port_Identifier of the SSP_Port to which the SCSI READ CAPACITY command shall be sent.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP target port discovered through the specified local port on the specified HBA. It shall have a value of 0 if no expander SMP target ports were discovered. It shall be ignored if the local port is not a SAS Port.

Argument **smhbaLUN** shall be the SCSI LUN to which the SCSI READ CAPACITY command shall be sent.

Argument **pRspBuffer** shall be a pointer to a buffer to receive the SCSI READ CAPACITY command response.

Argument **pRspBufferSize** shall be a pointer to the size in bytes of the buffer to receive the SCSI READ CAPACITY command response.

Argument **pScsiStatus** shall be a pointer to a buffer to receive SCSI status.

Argument **pSenseBuffer** shall be a pointer to a buffer to receive SCSI sense data.

Argument **pSenseBufferSize** shall be a pointer to the size in bytes of the buffer to receive SCSI sense data.

7.8.3.4 Return Values

The returned function value shall be as specified in table 64.

Table 64 — Returned Function Values for SMHBA_ScsiReadCapacity

Value	Function result
HBA_STATUS_OK	The complete payload of a reply to the SCSI READ CAPACITY command has been returned.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_NOT_A_TARGET	The remote end port identified by **discoveredPortWWN** does not have SCSI target port functionality.
HBA_STATUS_ERROR_TARGET_BUSY	Unable to send the requested command without causing a SCSI overlapped command condition.
HBA_STATUS_SCSI_CHECK_CONDITION	Returned SCSI status indicates a SCSI CHECK CONDITION.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the function value is HBA_STATUS_OK, the buffer pointed to by **pRspBuffer** shall contain the response to the SCSI READ CAPACITY command.

The value pointed to by **pRspBufferSize** shall be the size in bytes of the response returned by the command. This shall not exceed the size passed as an argument at this pointer.

The value pointed to by **pScsiStatus** shall be the SCSI status (see SAM-5). If the function value is HBA_STATUS_OK or HBA_STATUS_SCSI_CHECK_CONDITION, the value of the SCSI status may be interpreted based on the SAM-4 standard. A SCSI status of GOOD indicates a SCSI response is in the

buffer pointed to by **pRspBuffer**. A SCSI status of CHECK CONDITION indicates no value is stored in the buffer pointed to by **pRspBuffer**, and the buffer pointed to by **pSenseBuffer** shall contain failure information if available. All other SCSI status codes should be interpreted by reference to SAM-5.

If the function value is HBA_STATUS_SCSI_CHECK_CONDITION, the buffer pointed to by **pSenseBuffer** shall contain the sense data for the command.

The value pointed to by **pSenseBufferSize** shall be the size in bytes of the sense information returned by the command. It shall not exceed the size passed as an argument at this pointer.

7.8.4 SMHBA_ScsiManagementIn

7.8.4.1 Format

```
HBA_STATUS SMHBA_ScsiManagementIn(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN discoveredPortWWN,
       HBA_WWN domainPortWWN,
       SMHBA_SCSILUN smhbaLUN,
       HBA_UINT8 managementProtocol,
       HBA_UINT16 managementProtocolSpecific1
       HBA_UINT8 managementProtocolSpecific2
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize,
       HBA_UINT8 *pScsiStatus,
       void *pSenseBuffer,
       HBA_UINT32 *pSenseBufferSize
);
```

7.8.4.2 Description

The SMHBA_ScsiManagementIn function shall send a SCSI MANAGEMENT IN command (see SPC-4) to a SCSI logical unit through a remote FCP_Port or SSP_Port.

A SCSI command shall not be sent through a remote FCP_Port or SSP_Port that does not have SCSI target port functionality. An HBA API library shall not cause a SCSI overlapped command condition (see SAM-5). Proper use of tagged commands (see SAM-5) is an acceptable means of avoiding a SCSI overlapped command condition with SCSI logical units that support tagged commands.

The SCSI MANAGEMENT PROTOCOL IN command is used to retrieve management protocol information or the results of one or more SCSI MANAGEMENT PROTOCOL OUT commands. The contents of the managementProtocolSpecifc1 attribute and the managementProtocolSpecific2 attribute depend on the protocol specified by the managementProtocol attribute. The format of the data placed in the Response Buffer also depends on the protocol specified by the managementProtocol attribute.

The management protocol transported by the SCSI MANAGEMENT PROTOCOL IN command may have been originally designed to be transported by a different scheme that provides significantly better authentication of end points than is possible in a SCSI infrastructure. Designers of the SCSI “mapping” of such protocols should provide their own security features (including authentication) in the mapping, or else should limit the functionality supported in the SCSI case to read-only functions that do not affect the operation of the managed device.

7.8.4.3 Arguments

Argument **handle** shall be a handle to an HBA through which the SCSI MANAGEMENT IN command shall be sent.

Argument **hbaPortWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port on the specified HBA through which the SCSI MANAGEMENT IN command shall be sent.

Argument **discoveredPortWWN** shall be the Name_Identifier of the FCP_Port, or the Port_Identifier of the SSP_Port to which the SCSI MANAGEMENT IN command shall be sent.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP target port discovered through the specified local port on the specified HBA. It shall have a value of 0 if no expander SMP target ports were discovered. It shall be ignored if the local port is not a SAS Port.

Argument **smhbaLUN** shall be the SCSI LUN to which the SCSI MANAGEMENT IN command shall be sent.

Argument **managementProtocol** shall be the identifier of the Management Protocol being supported (see SPC-4).

Argument **managementProtocolSpecific1** shall be the information contained in the SCSI MANAGEMENT IN CDB field of the same name (see SPC-4).

Argument **managementProtocolSpecific2** shall be the information contained in the SCSI MANAGEMENT IN CDB field of the same name (see SPC-4).

Argument **pRspBuffer** shall be a pointer to a buffer to receive the SCSI MANAGEMENT IN command response.

Argument **pRspBufferSize** shall be a pointer to the size in bytes of the buffer to receive the SCSI MANAGEMENT IN command response.

Argument **pScsiStatus** shall be a pointer to a buffer to receive SCSI status.

Argument **pSenseBuffer** shall be a pointer to a buffer to receive SCSI sense data.

Argument **pSenseBufferSize** shall be a pointer to the size in bytes of the buffer to receive SCSI sense data.

7.8.4.4 Return Values

The returned function value shall be as specified in table 1.

Table 65 — Returned Function Values for SMHBA_ScsiManagementIn

Value	Function result
HBA_STATUS_OK	The complete payload of a reply to the SCSI MANAGEMENT IN command has been returned.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_NOT_A_TARGET	The remote end port identified by **discoveredPortWWN** does not have SCSI target port functionality.
HBA_STATUS_ERROR_TARGET_BUSY	Unable to send the requested command without causing a SCSI overlapped command condition.
HBA_STATUS_SCSI_CHECK_CONDITION	Returned SCSI status indicates a SCSI CHECK CONDITION.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the function value is HBA_STATUS_OK, the buffer pointed to by **pRspBuffer** shall contain the response to the SCSI MANAGEMENT IN command.

The value pointed to by **pRspBufferSize** shall be the size in bytes of the response returned by the command. This shall not exceed the size passed as an argument at this pointer.

The value pointed to by **pScsiStatus** shall be the SCSI status (see SAM-5). If the function value is HBA_STATUS_OK or HBA_STATUS_SCSI_CHECK_CONDITION, the value of the SCSI status may be interpreted based on the SAM-5 standard. A SCSI status of GOOD indicates a SCSI response is in the buffer pointed to by **pRspBuffer**. A SCSI status of CHECK CONDITION indicates no value is stored in the buffer pointed to by **pRspBuffer**, and the buffer pointed to by **pSenseBuffer** shall contain failure information if available. All other SCSI status codes should be interpreted by reference to SAM-5.

If the function value is HBA_STATUS_SCSI_CHECK_CONDITION, the buffer pointed to by **pSenseBuffer** shall contain the sense data for the command.

The value pointed to by **pSenseBufferSize** shall be the size in bytes of the sense information returned by the command. This shall not exceed the size passed as an argument at this pointer.

7.8.5 SMHBA_ScsiManagementOut

7.8.5.1 Format

```
HBA_STATUS SMHBA_ScsiManagementOut(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN discoveredPortWWN,
       HBA_WWN domainPortWWN,
       SMHBA_SCSILUN smhbaLUN,
       HBA_UINT8 managementProtocol,
       HBA_UINT16 managementProtocolSpecific1
       HBA_UINT8 managementProtocolSpecific2
       void *pDataOutBuffer,
       HBA_UINT32 DataOutBufferSize,
       HBA_UINT8 *pScsiStatus,
       void *pSenseBuffer,
       HBA_UINT32 *pSenseBufferSize
);
```

7.8.5.2 Description

The SMHBA_ScsiManagement Out function shall send a SCSI MANAGEMENT OUT command (see SPC-4) to a SCSI logical unit through a remote FCP_Port or SSP_Port.

A SCSI command shall not be sent through a remote FCP_Port or SSP_Port that does not have SCSI target port functionality. An HBA API library shall not cause a SCSI overlapped command condition (see SAM-5). Proper use of tagged commands (see SAM-5) is an acceptable means of avoiding a SCSI overlapped command condition with SCSI logical units that support tagged commands.

The SCSI MANAGEMENT PROTOCOL OUT command is used to send data to the logical unit. The data sent specifies one or more operations to be performed by the logical unit. The format and function of the operations depends on the contents of the managementProtocol argument. Depending on the protocol specified by the managementProtocol argument, the application client may use the SMHBA_ScsiManagementIn function to retrieve data derived from these operations.

The contents of the managementProtocolSpecific1attribute and the managementProtocolSpecific2 attribute depend on the protocol specified by the managementProtocol attribute. The format of the data placed in the Data Out Buffer also depends on the protocol specified by the managementProtocol attribute.

The management protocol transported by the SCSI MANAGEMENT PROTOCOL OUT may have been originally designed to be transported by a different scheme that provides significantly better authentication of end points than is possible in a SCSI infrastructure. Designers of the SCSI “mapping” of such protocols should provide their own security features (including authentication) in the mapping, or else should limit the functionality supported in the SCSI case to read-only functions that do not affect the operation of the managed device.

7.8.5.3 Arguments

Argument **handle** shall be a handle to an HBA through which the SCSI MANAGEMENT OUT command shall be sent.

Argument **hbaPortWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port on the specified HBA through which the SCSI MANAGEMENT OUT command shall be sent.

Argument **discoveredPortWWN** shall be the Name_Identifier of the FCP_Port, or the Port_Identifier of the SSP_Port to which the SCSI MANAGEMENT OUT command shall be sent.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP target port discovered through the specified local port on the specified HBA. It shall have a value of 0 if no expander SMP target ports were discovered. It shall be ignored if the local port is not a SAS Port.

Argument **smhbaLUN** shall be the SCSI LUN to which the SCSI MANAGEMENT OUT command shall be sent.

Argument **managementProtocol** shall be the identifier of the Management Protocol being supported (see SPC-4).

Argument **managementProtocolSpecific1** shall be the information contained in the SCSI MANAGEMENT IN CDB field of the same name (see SPC-4).

Argument **managementProtocolSpecific2** shall be the information contained in the SCSI MANAGEMENT IN CDB field of the same name (see SPC-4).

Argument **pDataOutBuffer** shall be a pointer to a buffer containing the SCSI MANAGEMENT OUT command data transfer.

Argument **DataOutBufferSize** shall be the size in bytes of the SCSI MANAGEMENT OUT command data transfer.

Argument **pScsiStatus** shall be a pointer to a buffer to receive SCSI status.

Argument **pSenseBuffer** shall be a pointer to a buffer to receive SCSI sense data.

Argument **pSenseBufferSize** shall be a pointer to the size in bytes of the buffer to receive SCSI sense data.

7.8.5.4 Return Values

The returned function value shall be as specified in table 2.

Table 66 — Returned Function Values for SMHBA_ScsiManagementOut

Value	Function result
HBA_STATUS_OK	The complete payload of a reply to the SCSI MANAGEMENT OUT command has been returned.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local FC_Port with Name_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access a local SAS Port with Port_Identifier identified by **hbaportWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
HBA_STATUS_ERROR_NOT_A_TARGET	The remote end port identified by **discoveredPortWWN** does not have SCSI target port functionality.
HBA_STATUS_ERROR_TARGET_BUSY	Unable to send the requested command without causing a SCSI overlapped command condition.
HBA_STATUS_SCSI_CHECK_CONDITION	Returned SCSI status indicates a SCSI CHECK CONDITION.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the function value is HBA_STATUS_OK, the data in ther buffer pointed to by **pRspBuffer** shall have been successfully transferred to the target device by the SCSI MANAGEMENT OUT command.

The value in **DataOutBufferSize** shall be unchanged.

The value pointed to by **pScsiStatus** shall be the SCSI status (see SAM-5). If the function value is HBA_STATUS_OK or HBA_STATUS_SCSI_CHECK_CONDITION, the value of the SCSI status may be interpreted based on the SAM-5 standard. A SCSI status of GOOD indicates that the data in the buffer pointed to by **pDataOutBuffer** has been successfully transferred. A SCSI status of CHECK CONDITION indicates that the buffer pointed to by **pSenseBuffer** shall contain failure information if available. All other SCSI status codes should be interpreted by reference to SAM-5.

If the function value is HBA_STATUS_SCSI_CHECK_CONDITION, the buffer pointed to by **pSenseBuffer** shall contain the sense data for the command.

The value pointed to by **pSenseBufferSize** shall be the size in bytes of the sense information returned by the command. It shall not exceed the size passed as an argument at this pointer.

7.9 Event Handling Functions

7.9.1 Overview of SM-HBA Event Reporting

The functions defined in this section support an asynchronous event reporting model.

7.9.1.1 Asynchronous Event Reporting Behavior Model

The asynchronous event reporting method provides a selective and prompt means of notifying interested applications of HBA-detected events. It comprises a set of functions that allow applications to register for notification of specified groups of events.

In the asynchronous event reporting method, an application shall be notified of an event occurrence by a callback to a function that has been registered by the application. The parameters of the callback function shall identify and characterize the event. The call shall occur after detection of the event; however, this standard places no specific limits on the timing of the call.

For the purpose of asynchronous event reporting, events detected by an HBA shall be grouped into event categories and into event types within categories. When an application registers a callback function for asynchronous event notification, it specifies selection criteria for events that shall be reported via that callback function. Selection criteria include the event category and the source of the event. Selection criteria for Port category events (see 6.11.1.4) also include the specific event type. Depending on the registration function, the source may be the local system, a local HBA, a local end port, FCP-4 target device, or an SSP target device.

An application that is registered for events shall be notified of every detected event that meets the selection criteria for a registered callback function. An application shall not be notified of any event that does not match the selection criteria for any callback function that is registered at the time of occurrence of the event.

An application that has registered for notification of events matching certain selection criteria may later deregister for notification of that category and source of events.

If an application registers a callback function matching certain selection criteria without explicitly deregistering previous callback functions for the same event selection criteria, each registered function shall be called on occurrence of any event matching those selection criteria.

Upon registration for statistical events, an application also specifies the conditions of statistical counters that shall be detected as an event.

An application may register for multiple groups of events with the same or differing callback functions. On registering for notification of a group of events, an application shall provide a void pointer that shall be passed to the callback. An application that registers multiple groups of events with the same callback function may use the data at that pointer to identify the registration call that enabled each callback.

Multiple applications may register concurrently for the same events. In this case, each event occurrence shall be reported to each registered application.

The arrival of an RSCN ELS shall cause a separate event for each Affected Port_ID Page carried by the RSCN.

7.9.1.2 Registration for Events with diverse HBA specific software

When an application calls an HBA API library function to register for asynchronous events, the HBA API library may in turn rely on some form of registration with HBA specific software. A wrapper library shall repeat the same event registration call to each HBA specific library. Within a single HBA API library, some HBA specific software may successfully process the registration, some may indicate it is unsupported, and some may fail to register for other reasons. In the presence of variant responses to event registration from HBA specific software, the behavior of the HBA API library shall be as follows:

a) the HBA API library shall continue to register with each instance of HBA specific software regardless of the response from any instance of HBA specific software;
b) if all instances of HBA specific software indicate the same result, the HBA API library shall return a status appropriate to that result;
c) if any instance of HBA specific software indicated successful registration, the HBA API library shall return HBA_STATUS_OK;
d) if any instance of HBA specific software indicated nonsupport for the event being registered and no instance of HBA specific software indicated successful registration, the HBA API library shall return HBA_STATUS_ERROR_NOT_SUPPORTED;
e) if no instance of HBA specific software indicated successful registration or nonsupport for the event being registered, but not all instances of HBA specific software indicated the same result, the HBA API library shall return a status appropriate to the result indicated by one of the instances of HBA specific software, chosen in a vendor specific manner; and
f) if not all instances of HBA specific software indicated the same result, the HBA API library shall follow the other rules in this subclause and in addition, for each instance of HBA specific software that indicated a result other than successful completion, the HBA API library shall make a nonvolatile record in a vendor specific manner of the identity of the function call and the instance of HBA specific software and the result it indicated.

NOTE 28 - It is suggested to use the stderr device on unix systems and the event log on Windows systems to make a nonvolatile record of event registration errors.

7.9.2 SMHBA_RegisterForAdapterAddEvents

7.9.2.1 Format

```
HBA_STATUS SMHBA_RegisterForAdapterAddEvents(
       void (*pCallback) (
              void *pData,
              HBA_WWN portWWN,
              HBA_UINT32 eventType
       ),
       void *pUserData,
       HBA_CALLBACKHANDLE *pCallbackHandle
       );
```

7.9.2.2 Description

The SMHBA_RegisterForAdapterAddEvents function shall register an application defined function that shall be called upon occurrence of HBA add category asynchronous events. When a new HBA is added to the local system, this callback shall be called with the Name_Identifier of any FC_Port, or the Port_Identifier of any SAS Port on the new HBA. The event type shall be HBA_EVENT_ADAPTER_ADD. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.2.3 Arguments

Argument **pCallback** shall be a pointer to the entry to the callback function.

Argument **pUserData** shall be a pointer that shall be passed to the callback function with each event. This may be used for correlating the event with the source of its event registration.

Argument **pCallbackHandle** shall be a pointer to a structure in which an identifier that may be used to deregister the callback may be returned.

7.9.2.4 Return Values

The returned function value shall be as specified in table 67.

Table 67 — Returned Function Values for SMHBA_RegisterForAdapterAddEvents

Value	Function result
HBA_STATUS_OK	The callback function was successfully registered.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the structure pointed to by **pCallbackHandle** shall contain an identifier that may be used to deregister the callback routine.

7.9.2.5 Callback Arguments

Argument **pData** shall be the pointer that was passed from registration for this event category. This may be used for correlating the event with the source of its event registration.

Argument **portWWN** shall be the Name_Identifier of any of the FC_Port, or the Port_Identifier of any of the SAS Port on the HBA that was added.

Argument **eventType** shall be HBA_EVENT_ADAPTER_ADD.

7.9.3 SMHBA_RegisterForAdapterEvents

7.9.3.1 Format

```
HBA_STATUS SMHBA_RegisterForAdapterEvents(
       void (*pCallback) (
              void *pData,
              HBA_WWN portWWN,
              HBA_UINT32 eventType
       ),
       void *pUserData,
       HBA_HANDLE handle,
       HBA_CALLBACKHANDLE *pCallbackHandle
);
```

7.9.3.2 Description

The SMHBA_RegisterForAdapterEvents function shall register an application defined function that shall be called upon occurrence of HBA category asynchronous events. When an HBA category event occurs for the specified HBA, the callback function shall be called with event type of HBA_EVENT_ADAPTER_REMOVE or HBA_EVENT_ADAPTER_CHANGE. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.3.3 Arguments

Argument **pCallback** shall be a pointer to the entry to the callback function.

Argument **pUserData** shall be a pointer that shall be passed to the callback function with each event. This may be used for correlating the event with the source of its event registration.

Argument **handle** shall be a handle to an HBA for which event callbacks are requested.

Argument **pCallbackHandle** shall be a pointer to a structure in which an identifier that may be used to deregister the callback may be returned.

7.9.3.4 Return Values

The returned function value shall be as specified in Table 68.

Table 68 — Returned Function Values for SMHBA_RegisterForAdapterEvents

Value	Function result
HBA_STATUS_OK	The callback function was successfully registered.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the structure pointed to by **pCallbackHandle** shall contain an identifier that may be used to deregister the callback routine.

7.9.3.5 Callback Arguments

Argument **pData** shall be the pointer that was passed from registration for this event category. This may be used for correlating the event with the source of its event registration.

Argument **portWWN** shall be the Name_Identifier of any FC_Port, or the Port_Identifier of any SAS Port on the specified HBA that detected the event. The client should re-discover all aspects of the HBA and ALL connected FC_Ports and SAS Ports as the prior state may not be accurate.

Argument **eventType** shall be a value specified in 6.11.1.3 indicating the type of event that occurred.

7.9.4 SMHBA_RegisterForAdapterPortEvents

7.9.4.1 Format

```
HBA_STATUS SMHBA_RegisterForAdapterPortEvents(
       void (*pCallback) (
              void *pData,
```

```
            HBA_WWN portWWN,
            HBA_UINT32 eventType,
            HBA_UINT32 fabricPortID
        ),
        void *pUserData,
        HBA_HANDLE handle,
        HBA_WWN portWWN,
        HBA_UINT32 specificEventType,
        HBA_CALLBACKHANDLE *pCallbackHandle
);
```

7.9.4.2 Description

The SMHBA_RegisterForAdapterPortEvents function shall register an application defined function that shall be called upon occurrence of specific port category asynchronous events. When a port category event occurs for the specified port, the callback function is called with event type set to the appropriate event.

The application may register a call back function for one or more port event types specified in 6.11.1.4. If the selected event is of type HBA_EVENT_PORT_ALL, then the application defined function shall be called upon occurrence of any of the events specified in 6.11.1.4.

NOTE 29 - An application may invoke SMHBA_RegisterForAdapterPortEvents more than once to register the same call back function for different port category asynchronous events.

If the event is of type HBA_EVENT_PORT_FABRIC, the callback argument fabricPortID shall contain the RSCN affected Port ID page for the sub-section of the fabric that has changed (see the RSCN definition in FC-LS-2). The arrival of an RSCN ELS shall cause a separate event for each Affected Port ID Page carried by the RSCN. For all other event types, fabricPortID shall be ignored. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.4.3 Arguments

Argument **pCallback** shall be a pointer to the entry to the callback function.

Argument **pUserData** shall be a pointer that shall be passed to the callback function with each event. This may be used for correlating the event with the source of its event registration.

Argument **handle** shall be a handle to an HBA for which event callbacks are requested.

Argument **portWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port on the specified HBA for which callbacks are requested.

Argument **specificEventType** shall be a value specified in 6.11.1.4 indicating the type of event for which registration is requested, or shall be HBA_EVENT_PORT_ALL to request registration for all Port category events.

Argument **pCallbackHandle** shall be a pointer to a structure in which an identifier that may be used to deregister the callback may be returned.

7.9.4.4 Return Values

The returned function value shall be as specified in Table 69.

Table 69 — Returned Function Values for SMHBA_RegisterForAdapterPortEvents

Value	Function result
HBA_STATUS_OK	The callback function was successfully registered.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local FC_Port with Name_Identifier identified by **portWWN**, or the HBA identified by **handle** is not able to access a local SAS Port with Port_Identifier identified by **portWWN.**
HBA_STATUS_ERROR_BAD_EVENT_TYPE	The event type referenced by **specificEventType** is not valid.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the structure pointed to by **pCallbackHandle** shall contain an identifier that may be used to deregister the callback routine.

7.9.4.5 Callback Arguments

Argument **pData** shall be the pointer that was passed from registration for this event category. This may be used for correlating the event with the source of its event registration.

Argument **portWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port on the specified HBA that detected the event.

Argument **eventType** shall be a value specified in 6.11.1.4 indicating the type of event that occurred.

Argument **fabricPortID** shall contain the RSCN affected Port ID page, as per the RSCN definition in FC-LS-2 if the event is of type HBA_EVENT_PORT_FABRIC. For all other event types, **fabricPortID** shall be ignored.

7.9.5 SMHBA_RegisterForAdapterPortStatEvents

7.9.5.1 Format

```
HBA_STATUS SMHBA_RegisterForAdapterPortStatEvents(
       void (*pCallback) (
              void *pData,
              HBA_WWN portWWN,
              HBA_UINT32 protocolType,
              HBA_UINT32 eventType,
       ),
       void *pUserData,
       HBA_HANDLE handle,
       HBA_WWN portWWN,
```

```
        HBA_UINT32  protocolType,
        SMHBA_PROTOCOLSTATISTICS stats,
        HBA_UINT32 statType,
        HBA_CALLBACKHANDLE *pCallbackHandle
);
```

7.9.5.2 Description

The SMHBA_RegisterForAdapterPortStatEvents function shall define conditions causing an HBA port statistics category asynchronous event and register an application defined function that shall be called upon occurrence of the conditions that are defined. This may be used for statistic threshold crossing or growth rate events. Multiple statistics may be registered in one call by setting more than one statistic in the stats argument to a non-zero value. For threshold events, once a specific threshold is crossed, the callback shall be automatically de-registered for that statistic. If other statistics were registered for that callback, they shall remain in effect until they are crossed. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.5.3 Arguments

Argument **pCallback** shall be a pointer to the entry to the callback function.

Argument **pUserData** shall be a pointer that shall be passed to the callback function with each event. This may be used for correlating the event with the source of its event registration.

Argument **handle** shall be a handle to an HBA for which event callbacks are requested.

Argument **portWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port on the specified HBA for which event callbacks are requested.

Argument **protocolType** shall be the data structure TYPE (see FC-FS-3) for FC-4, or the SAS protocol type (see) to monitor for statistics on the specified port for which event call backs are requested.

Argument **stats** shall be a SMHBA_PROTOCOLSTATISTICS structure in which nonzero values shall indicate the counters to be monitored. If **statType** is HBA_EVENT_PORT_STAT_THRESHOLD, any non-null values in the **stats** structure shall be interpreted as the thresholds to monitor. If **statType** is HBA_EVENT_PORT_STAT_GROWTH, any non-null values in the **stats** structure shall be interpreted as growth rate numbers over one minute, although the frequency at which the growth is monitored is vendor specific.

Argument **statType** shall be a value specified in 6.11.1.6 that shall determine whether the events registered by this call are threshold crossing or growth rate of the indicated counters.

Argument **pCallbackHandle** shall be a pointer to a structure in which an identifier that may be used to deregister the callback may be returned.

7.9.5.4 Return Values

The returned function value shall be as specified in Table 70.

Table 70 — Returned Function Values for SMHBA_RegisterForAdapterPortStatEvents

Value	Function result
HBA_STATUS_OK	The callback function was successfully registered.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local FC_Port with Name_Identifier identified by **portWWN**, or the HBA identified by **handle** is not able to access a local SAS Port with Port_Identifier identified by **portWWN.**
HBA_STATUS_ERROR_INVALID_PROTOCOL_TYPE	The protocol referenced by **protocolType** is not valid.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA specific library or underlying system does not support statistic events, or the HBA referenced by **handle** does not support this function.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the structure pointed to by **pCallbackHandle** shall contain an identifier that may be used to deregister the callback routine.

7.9.5.5 Callback Arguments

Argument **pData** shall be the pointer that was passed from registration for this event category. This may be used for correlating the event with the source of its event registration.

Argument **portWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port on the specified HBA that detected the event.

Argument **protocolType** shall be the data structure TYPE (see FC-FS-3) for FC-4, or the SAS protocol type (see) that detected the event.

Argument **eventType** shall be a value specified in 6.11.1.5 indicating the type of event that occurred.

7.9.6 SMHBA2_RegisterForAdapterPhyStatEvents

7.9.6.1 Format

```
HBA_STATUS SMHBA2_RegisterForAdapterPhyStatEvents(
       void (*pCallback) (
              void *pData,
              HBA_HANDLE phyHandle,
              HBA_UINT32 eventType,
```

```
        ),
        void *pUserData,
        HBA_HANDLE phyHandle,
        SMHBA2_PHYSTATISTICS stats,
        HBA_UINT32 statType,
        HBA_CALLBACKHANDLE *pCallbackHandle
);
```

7.9.6.2 Description

The SMHBA2_RegisterForAdapterPhyStatEvents function shall define conditions causing an HBA phy statistics category asynchronous event and register an application defined function that shall be called upon occurrence of the conditions that are defined. This may be used for statistic threshold crossing or growth rate events. Multiple statistics may be registered in one call by setting more than one statistic in the stats argument to a non-zero value. For threshold events, once a specific threshold is crossed, the callback shall be automatically de-registered for that statistic. If other statistics were registered for that callback, they shall remain in effect until they are crossed. To terminate event delivery, HBA_RemoveCallback shall be called.

NOTE 30 - The management client application is assumed to have prior knowledge of the phy type attribute of the phy for which conditions are being defined for event generation. In addition, the management client application shall invoke SMHBA2_RegisterForAdapterPhyStatEvents with appropriate stats arguments relevant to the phy type.

7.9.6.3 Arguments

Argument **pCallback** shall be a pointer to the entry to the callback function.

Argument **pUserData** shall be a pointer that shall be passed to the callback function with each event. This may be used for correlating the event with the source of its event registration.

Argument **phyHandle** shall be the handle of the phy on the identified port for which callbacks are requested.

Argument **stats** shall be a SMHBA2_PHYSTATISTICS structure in which nonzero values shall indicate the counters to be monitored. If **statType** is HBA_EVENT_PHY_STAT_THRESHOLD, any non-null values in the **stats** structure shall be interpreted as the thresholds to monitor. If **statType** is HBA_EVENT_PHY_STAT_GROWTH, any non-null values in the **stats** structure shall be interpreted as growth rate numbers over one minute, although the frequency at which the growth is monitored is vendor specific.

Argument **statType** shall be a value specified in 6.11.1.6 that shall determine whether the events registered by this call are threshold crossing or growth rate of the indicated counters.

Argument **pCallbackHandle** shall be a pointer to a structure in which an identifier that may be used to deregister the callback may be returned.

7.9.6.4 Return Values

The returned function value shall be as specified in table 71.

Table 71 — Returned Function Values for SMHBA2_RegisterForAdapterPhyStatEvents

Value	Function result
HBA_STATUS_OK	The callback function was successfully registered.
HBA_STATUS_ERROR_INVALID_HANDLE	**phyHandle** in not a phy handle
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA specific library or underlying system does not support statistic events.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the structure pointed to by **pCallbackHandle** shall contain an identifier that may be used to deregister the callback routine.

7.9.6.5 Callback Arguments

Argument **pData** shall be the pointer that was passed from registration for this event category. This may be used for correlating the event with the source of its event registration.

Argument **phyHandle** shall be the handle of the phy that detected the event.

Argument **eventType** shall be a value specified in 6.11.1.6 indicating the type of event that occurred.

7.9.7 SMHBA_RegisterForTargetEvents

7.9.7.1 Format

```
HBA_STATUS SMHBA_RegisterForTargetEvents(
       void (*pCallback) (
              void *pData,
              HBA_WWN hbaPortWWN,
              HBA_WWN discoveredPortWWN,
              HBA_WWN domainPortWWN,
              HBA_UINT32 eventType,
       ),
       void *pUserData,
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN discoveredPortWWN,
       HBA_WWN domainPortWWN,
       HBA_CALLBACKHANDLE *pCallbackHandle,
       HBA_UINT32 allTargets
);
```

7.9.7.2 Description

The SMHBA_RegisterForTargetEvents function shall register an application defined function that shall be called upon occurrence of target category asynchronous events. When an event concerning an FCP-4 target port or SSP target port occurs, the callback function shall be called with event type of HBA_EVENT_TARGET_OFFLINE, HBA_EVENT_TARGET_ONLINE, HBA_EVENT_TARGET_REMOVED, or HBA_EVENT_TARGET_UNKNOWN. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.7.3 Arguments

Argument **pCallback** shall be a pointer to the entry to the callback function.

Argument **pUserData** shall be a pointer that shall be passed to the callback function with each event. This may be used for correlating the event with the source of its event registration.

Argument **handle** shall be a handle to an HBA for which event callbacks are requested.

Argument **hbaportWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port on the specified HBA for which event callbacks are requested.

Argument **discoveredPortWWN** shall be the Name_Identifier of the FCP-4 target port, or the Port_Identifier of the SSP target port for which event callbacks are requested.

Argument **domainPortWWN** shall be the Port_Identifier of any expander SMP target port discovered through the specified local port on the specified HBA. It shall have a value of 0 if no SMP target ports were discovered. It shall be ignored if the local port is an FC_Port.

Argument **pCallbackHandle** shall be a pointer to a structure in which an identifier that may be used to deregister the callback may be returned.

Argument **allTargets** shall indicate the scope of target ports registered by this call. If **allTargets** is set to a non-zero value, the value in **discoveredPortWWN** shall be ignored, and the call shall register events for all current and future discovered FCP-4 target ports, or SSP target ports. If **allTargets** is set to zero, only event for the FCP-4 target port, or SSP target port specified by **discoveredPortWWN** shall be registered by this call.

7.9.7.4 Return Values

The returned function value shall be as specified in table 72.

Table 72 — Returned Function Values for SMHBA_RegisterForTargetEvents

Value	Function result
HBA_STATUS_OK	The callback function was successfully registered.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The **domainPortWWN** is zero and the specified HBA has access to more than one SAS Port with the specified Port_Identifier.
HBA_STATUS_ERROR_ILLEGAL_WWN	The HBA identified by **handle** is not able to access a local FC_Port with Name_Identifier identified by **hbaPortWWN**, or the HBA identified by **handle** is not able to access a local SAS Port with Port_Identifier identified by **hbaPortWWN,** or the HBA identified by **handle** is not able to access a discovered port identified by **discoveredPortWWN**, or the HBA identified by **handle** is not able to access an expander SMP target port identified by **domainPortWWN**.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the structure pointed to by **pCallbackHandle** shall contain an identifier that may be used to deregister the callback routine.

7.9.7.5 Callback Arguments

Argument **pData** shall be the pointer that was passed from registration for this event category. This may be used for correlating the event with the source of its event registration.

Argument **hbaportWWN** shall be the Name_Identifier of the FC_Port, or the Port_Identifier of the SAS Port through which the target event was detected.

Argument **discoveredPortWWN** shall be the Name_Identifier of the FCP-4 target port, or the Port_Identifier of the SSP target port at which the target event was deteced.

Argument **domainPortWWN** shall be the Port_Identifier of the expander SMP target port with the smallest Port_Identifier that was discovered on the domain in which the target event was detected. The size of a Port_Identifier shall be determined by byte-by-byte numeric comparison. It shall be ignored if the local port is an FC_Port.

Argument **eventType** shall be a value specified in 6.11.1.7 indicating the type of event that occurred.

7.9.8 HBA_RegisterForLinkEvents

7.9.8.1 Format

```
HBA_STATUS HBA_RegisterForLinkEvents(
       void (*pCallback) (
              void *pData,
              HBA_WWN adapterWWN,
              HBA_UINT32 eventType,
              void *pRLIRBuffer,
              HBA_UINT32 RLIRBufferSize
       ),
       void *pUserData,
       void *pRLIRBuffer,
       HBA_UINT32 RLIRBufferSize,
       HBA_HANDLE handle,
       HBA_CALLBACKHANDLE *pCallbackHandle,
);
```

7.9.8.2 Description

The HBA_RegisterForLinkEvents function shall register an application defined function that shall be called upon occurrence of link category asynchronous events on a specified HBA. When an event concerning a fabric link is detected by the HBA, the callback function shall be called. Arrival of an RLIR ELS shall be the only fabric link event type. Upon arrival of an RLIR ELS, the HBA or its driver shall provide ELS acknowledgement. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.8.3 Arguments

Argument **pCallback** shall be a pointer to the entry to the callback function.

Argument **pUserData** shall be a pointer that shall be passed to the callback function with each event. This may be used for correlating the event with the source of its event registration.

Argument **pRLIRBuffer** shall be a pointer to buffer in which RLIR data may be passed to the callback function. This buffer shall be overwritten at each entry to a fabric link event callback function that references it. It shall not be overwritten during the time between an entry to the callback function and its subsequent exit.

Argument **RLIRBufferSize** shall be the size in bytes of the buffer that **pRLIRBuffer** addresses.

Argument **handle** shall be a handle to an HBA for which event callbacks are requested.

Argument **pCallbackHandle** shall be a pointer to a structure in which an identifier that may be used to deregister the callback may be returned.

7.9.8.4 Return Values

The returned function value shall be as specified in table 73.

Table 73 — Returned Function Values for HBA_RegisterForLinkEvents

Value	Function result
HBA_STATUS_OK	The callback function was successfully registered.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA specific library or underlying system does not support link events.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

If the returned function value is HBA_STATUS_OK, the structure pointed to by **pCallbackHandle** shall contain an identifier that may be used to deregister the callback routine.

7.9.8.5 Callback Arguments

Argument **pData** shall be the pointer that was passed from registration for this event category. This may be used for correlating the event with the source of its event registration.

Argument **adapterWWN** shall be the N_Port_Name of any end port on the HBA from which a fabric link event is being reported.

Argument **eventType** shall be a value specified in 6.11.2.1 indicating the type of event that occurred. If it is HBA_EVENT_LINK_INCIDENT, an RLIR has arrived, and further information shall be provided in the data at **RLIRBuffer**. If it is HBA_EVENT_LINK_UNKNOWN, a fabric link or topology change was detected by means other than RLIR, and the data at **RLIRBuffer** shall be ignored.

Argument **pRLIRBuffer** shall be the pointer to the RLIR data buffer passed as an argument to the registration call. The buffer to which it points shall contain the payload data from the RLIR ELS being reported (see FC-LS-2). If the actual RLIR payload exceeds the size of the buffer originally registered, trailing data shall be truncated to the size specified as an argument on the original registration call.

Argument **RLIRBufferSize** shall be the size in bytes of the complete payload of the RLIR ELS. IF it exceeds the size specified as an argument on the original registration call, this shall indicate the returned data has been truncated to the size specified as an argument on the original registration call.

7.9.9 HBA_RemoveCallback

7.9.9.1 Format

```
HBA_STATUS HBA_RemoveCallback(
       HBA_CALLBACKHANDLE callbackHandle
);
```

7.9.9.2 Description

The HBA_RemoveCallback function shall remove an instance of a callback function specified by the identifier **callbackHandle**.

7.9.9.3 Arguments

Argument **callbackHandle** shall be the identifier returned by the asynchronous event registration function that shall be deregistered.

7.9.9.4 Return Values

The returned function value shall be as specified in table 74.

Table 74 — Returned Function Values for HBA_RemoveCallback

Value	Function result
HBA_STATUS_OK	The callback function was successfully removed.
any value in subclause 6.2	Any condition not covered elsewhere in this table. The return value may be HBA_STATUS_ERROR if no more specific condition is detected by the function.

8 Configuration

8.1 Overview

This clause applies only to HBA API libraries with OS independent structure. No part of this clause shall apply to HBA API libraries with OS specific structure. This clause specifies a uniform, complete, and persistent inventory of the components that compose an HBA API on a system and their relationships to one another. This clause facilitates both configuration services and HBA API implementations. This clause refers to features of specific operating systems.

A given environment using the HBA API may be the result of several installation processes from various implementors. Each implementor's installation process may install one or more HBA specific libraries and may install a version of the wrapper library. The process of installing a version of the wrapper library should include the preservation of any previously installed version so that it may be restored if necessary.

8.2 Win32

In a Win32 environment (e.g., Window NT, Windows 2000) the method for registering multiple vendors' HBA specific libraries shall be:

a) under the Registry, an HBA vendor shall install a registry key to indicate where the vendor library is installed. The registry key for an HBA specific library that is compliant with this standard shall be of the format

\HKEY_LOCAL_MACHINE\SOFTWARE\SNIA\SMHBA2*vendorid*

where *vendorid* in the key name shall be the reversed domain name of the vendor followed by "." followed by the vendor specific name for the library that uniquely identifies the vendor library, and

b) the key shall have a value named LibraryFile of type REG_SZ that contains the full path to the vendor's library.

Example:

\\HKEY_LOCAL_MACHINE\SOFTWARE\SNIA\SMHBA2\org.snia.sample
LibraryFile = "c:\Program Files\Samplevendor\Library.dll"

An HBA specific library that is compliant with this standard and one or more prior versions of this standard shall be installed with keys for all versions with which it is compliant.

The method used to load multiple vendors' libraries in a Win32 environment shall include these procedures:

a) a wrapper library that is compliant with this standard shall be installed as files named SMHBA2API.DLL and SMHBA2API.LIB and shall be installed in directory %systemroot%\System32\;
b) a wrapper library that is compliant with this standard and one or more prior versions of this standard shall be installed multiple times, once in accord with the file naming conventions for each version with which it is compliant;
c) the wrapper library shall read the registry to discover HBA specific library names;
d) using the Win32 routines LoadLibrary and GetProcAddress, the wrapper shall open and discover the appropriate vendors libraries;
e) the wrapper library shall use these libraries to discover the aggregate number of adapters and report this to the upper level application;
f) the names of the lower level adapters shall be passed through the wrapper library;

g) a call to open an HBA shall be switched by the wrapper library, which shall assign and use the upper 16 bits of the HBA_HANDLE to determine which HBA to address on a given routine; and
h) remaining calls shall be routed by the wrapper library to the appropriate HBA specific library given the HBA_HANDLE.

8.3 Unix

In a Unix environment the method for registering multiple vendors' HBA specific libraries uses library directory files. An HBA specific library that is compliant with this standard shall be registered by:

a) if it does not already exist, a text file /etc/smhba2.conf shall be created to serve as the library directory file; and
b) In the file /etc/smhba2.conf, an HBA vendor shall insert a text line to indicate where the vendor library is installed. The text line shall be of the format

vendorid<sp>vslpath

where:

A) *vendorid* in the key name shall be the reversed domain name of the vendor followed by "." followed by the vendor specific name for the library that uniquely identifies the vendor library;
B) *<sp>* shall be any nonnull combination of space characters and tab characters; and
C) *vslpath* shall be the full path to the HBA specific library.

Examples:

org.T11.sample /usr/lib/libhbaapi-reference-vsl.so
com.hotbiscuitsadapters.supervsl /usr/lib/sparcv9/lib-hba-supervsl.so

An HBA specific library that is compliant with this standard and one or more prior versions of this standard shall be registered multiple time, once in accord with the registration procedure for each version with which it is compliant.

The method used to load multiple vendors' libraries in a unix environment shall include the procedures in the following list:

a) a wrapper library that is compliant with this standard shall be installed as files named SMHBA2API.DLL and SMHBA2API.LIB and shall be installed in the directory appropriate to the library type:
A) 32-bit: /usr/lib; or
B) 64-bit: vendor-specific subdirectory of /usr/lib/ for 64-bit libraries (e.g., /usr/lib/sparcv9/);
b) a wrapper library that is compliant with this standard and one or more prior versions of this standard shall be installed multiple times, once in accord with the file naming conventions for each version with which it is compliant;
c) the wrapper library shall read the /etc/smhba2.conf file to discover the names of HBA specific libraries compliant with this standard;
d) using OS HBA specific library loading routines (e.g., dlopen and dlsym on Solaris), the wrapper shall open and discover the appropriate vendors libraries;
e) the wrapper library shall use these libraries to discover the aggregate number of adapters and report this to the upper level application;
f) the names of the lower level adapters shall be passed through the wrapper library;
g) a call to open an HBA shall be switched by the wrapper library, which shall assign and use the upper 16 bits of the HBA_HANDLE to determine which HBA to address on a given routine; and
h) remaining calls shall be routed by the wrapper library to the appropriate HBA specific library given the HBA_HANDLE.

Annex A
(Normative)

SM-HBA-2 Compliance Requirements

A.1 Overview

Software compliant with this standard shall observe all the normative specifications in the body of this standard, however, compliance does not require implementation of all specified features. This normative annex identifies the features that software compliant with this standard shall implement and how it shall indicate optional features that it does not implement.

Functions shall be mandatory, optional, or not allowed.

A compliant HBA specific library shall:

a) for each mandatory function that is associated with a supported port type (e.g., Fibre Channel and SAS), provide an entry point as specified in this standard that when entered shall return the response as specified in this standard;
b) for each optional function and for each mandatory function that is associated with an unsupported port type, provide an entry point as specified in this standard that when entered shall have the effects and return the response as specified in this standard or shall have no effect and return HBA_STATUS_ERROR_NOT_SUPPORTED; and
c) for a function that is not allowed, provide no entry point.

A compliant HBA API library, whether of OS specific or OS independent (i.e., wrapper library) structure, shall:

a) for each mandatory function, provide an entry point as specified in this standard; and
b) for a function that is not allowed, provide no entry point.

NOTE 31 - There are no optional functions for an HBA API library.

A compliant HBA API library that is a wrapper library shall:

a) for any function that identifies a specific HBA, call the appropriate HBA specific library function and return the response returned by the HBA specific library function; and
b) for any function that does not identify a specific HBA, perform the functions and return the response as specified in this standard.

A compliant HBA API library that is of OS specific structure shall:

a) for any function that identifies a specific HBA and for which the HBA specific software enables the function, perform the functions and return the response as specified in this standard;
b) for any function that identifies a specific HBA and for which the HBA specific software does not enable the function, return HBA_STATUS_ERROR_NOT_SUPPORTED; and
c) for any function that does not identify a specific HBA perform the functions and return the response as specified in this standard.

Attributes and statistics shall be mandatory, optional, or not allowed. Compliant software shall:

a) for each mandatory attribute and statistic, implement it as specified in this standard;
b) for each optional attribute and statistic, either implement it as specified in this standard or provide the value indicating unspecified; and
c) for each not allowed attribute and statistic, provide the value indicating unspecified.

A.2 Functions

Table A.1 specifies the requirements for implementing functions for software that is compliant with this standard. Different requirements are specified for an SM-HBA-2 API library than for SM-HBA-2 specific libraries. Functions not listed in this section are prohibited unless they are specified by a prior version of this standard to which the software is also compliant (e.g., SM-HBA, FC-HBA, and FC-MI).

Table A.1 — General Function Requirements (part 1 of 4)

Function	For SM-HBA-2 Library	For SM-HBA-2 specific libraries	Reference
Library Control Functions			
SMHBA2_GetVersion	M	M	7.2.1
HBA_LoadLibrary	M	M	7.2.2
HBA_FreeLibrary	M	M	7.2.3
SMHBA2_RegisterLibrary	N	M	7.2.4
SMHBA_GetWrapperLibraryAttributes	M	N	7.2.5
SMHBA_GetVendorLibraryAttributes	M	M	7.2.6
HBA_GetNumberOfAdapters	M	M	7.2.7
Object Attribute Functions			
SMHBA2_GetAdapterHandleByIndex	M	M	7.3.1
SMHBA2_GetAdapterAttributes	M	M	7.3.2
SMHBA2_GetAdapterBusAttributes	M	O	7.3.4
SMHBA_GetNumberofPorts	M	M	7.3.3
SMHBA2_GetPortType	M	M	7.3.5
SMHBA2_GetPortAttributes	M	M	7.3.6
SMHBA2_GetPortAttributesByWWN	M	O	7.3.7
SMHBA2_GetPhyType	M	M	7.3.8

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

Table A.1 — General Function Requirements (part 2 of 4)

Function	For SM-HBA-2 Library	For SM-HBA-2 specific libraries	Reference
SMHBA2_GetPhyAttributes	M	M	7.3.9
SMHBA2_GetPhyCtlrAttributes	M	M	7.3.10
SMHBA2_GetFabricInfo	M	M	7.3.11
Object Relationship Functions			
SMHBA2_GetPortsOnAdapter	M	M	7.4.1
SMHBA2_GetAdapterForPort	M	M	7.4.2
SMHBA2_GetLEPForPort	M	M	7.4.3
SMHBA2_GetDiscoveredPorts	M	M	7.4.4
SMHBA2_GetPhysOnAdapter	M	M	7.4.5
SMHBA2_GetAdapterForPhy	M	M	7.4.6
SMHBA2_GetPortsOnPhy	M	M	7.4.7
SMHBA2_GetPhysForPort	M	M	7.4.8
SMHBA2_GetCtlrForPhy	M	M	7.4.9
SMHBA2_GetPhyForCtlr	M	M	7.4.10
SMHBA2_GetFabricsForCtlr	M	M	7.4.11
SMHBA2_GetCtlrsForFabric	M	M	7.4.12
SMHBA2_GetFabricForPort	M	M	7.4.13
SMHBA2_GetPortsForFabric	M	M	7.4.14
Statistics Functions			
SMHBA2_GetPortStatistics	M	M	7.5.1
SMHBA2_GetProtocolStatistics	M	O	7.5.2
SMHBA2_GetPhyStatistics	M	M	7.5.3

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

Table A.1 — General Function Requirements (part 3 of 4)

Function	For SM-HBA-2 Library	For SM-HBA-2 specific libraries	Reference
SMHBA2_GetFIPStatistics	M	O	7.5.4
Target Information Functions			
SMHBA_GetBindingCapability	M	O	7.7.1
SMHBA_GetBindingSupport	M	O	7.7.2
SMHBA_SetBindingSupport	M	O	7.7.3
SMHBA_GetTargetMapping	M	O	7.7.4
SMHBA_GetPersistentBinding	M	O	7.7.5
SMHBA_SetPersistentBinding	M	O	7.7.6
SMHBA_RemovePersistentBindings	M	O	7.7.7
SMHBA_RemoveAllPersistentBindings	M	O	7.7.8
SMHBA_GetLUNStatistics	M	O	7.7.9
SCSI Information Functions			
SMHBA_ScsiInquiry	M	O	7.8.1
SMHBA_ScsiReportLuns	M	O	7.8.2
SMHBA_ScsiReadCapacity	M	O	7.8.3
Fabric and Domain Management Functions			
HBA_SendCTPassThruV2	M	O	7.6.1
HBA_SetRNIDMgmtInfo	M	O	7.6.2
HBA_GetRNIDMgmtInfo	M	O	7.6.3
HBA_SendRNIDV2	M	O	7.6.4
HBA_SendSRL	M	O	7.6.5
HBA_SendLIRR	M	O	7.6.6

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

Table A.1 — General Function Requirements (part 4 of 4)

Function	For SM-HBA-2 Library	For SM-HBA-2 specific libraries	Reference
HBA_SendRLS	M	O	7.6.7
SMHBA_SendTEST	M	O	7.6.8
SMHBA_SendECHO	M	O	7.6.9
SMHBA_SendSMPPassThru	M	O	7.6.10
Event Handling Functions			
SMHBA_RegisterForAdapterAddEvents	M	O	7.9.2
SMHBA_RegisterForAdapterEvents	M	O	7.9.3
SMHBA_RegisterForAdapterPortEvents	M	O	7.9.4
SMHBA_RegisterForAdapterPortStatEvents	M	O	7.9.5
SMHBA2_RegisterForAdapterPhyStatEvents	M	O	7.9.6
SMHBA_RegisterForTargetEvents	M	O	7.9.7
HBA_RegisterForLinkEvents	M	O	7.9.8
HBA_RemoveCallback	M	O	7.9.9

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.3 Generic Adapter Attributes

Table A.2 specifies the requirements for implementing Generic Adapter attributes for software that is compliant with this standard.

Table A.2 — Generic Adapter attributes

Attribute Name	Requirement	Value indicating unspecified	Reference
HBAHandle	M	not applicable	6.3.1.3.1
HBAOptions	M	not applicable	6.3.1.3.2
Manufacturer	M	not applicable	6.3.1.3.3
SerialNumber	M	not applicable	6.3.1.3.4
Model	M	not applicable	6.3.1.3.5
ModelDescription	O	Null string	6.3.1.3.6
HardwareVersion	O	Null string	6.3.1.3.7
DriverVersion	O	Null string	6.3.1.3.8
OptionROMVersion	O	Null string	6.3.1.3.9
FirmwareVersion	O	Null string	6.3.1.3.10
VendorSpecificID	O	zero	6.3.1.3.11
DriverName	O	Null string	6.3.1.3.12
HBASymbolicName	O	Null string	6.3.1.3.13
RedundantOptionROMVersion	O	Null string	6.3.1.3.14
RedundantFirmwareVersion	O	Null string	6.3.1.3.15

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.4 Generic Bus Attributes

Table A.3 specifies the requirements for implementing Generic Bus attributes for software that is compliant

with this standard.

Table A.3 — Generic Bus Attriutes

Attribute Name	Requirement	Value indicating unspecified	Reference
Type	M	not applicable	6.4.1.3.1
Address	M	not applicable	6.4.1.3.2

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.5 PCI Bus Attributes

Table A.4 specifies the requirements for implementing PCI Bus attributes for software that is compliant with this standard.

Table A.4 — PCI Bus Attributes

Attribute Name	Requirement	Value indicating unspecified	Reference
BusNumber	M	not applicable	6.4.2.3.1
DeviceNumber	M	not applicable	6.4.2.3.2
FunctionNumber	M	not applicable	6.4.2.3.3

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.6 Generic Port Attributes

Table A.5 specifies the requirements for implementing Generic Port Attributes for software that is compliant with this standard. Different requirements are specified for attributes of local end ports than for discovered ports.

Table A.5 — Generic Port Attributes

Attribute Name	For local end ports	For discovered ports	Value indicating unspecified	Reference
PortHandle	M	M	not applicable	6.5.1.3.1
PortType	M	O	HBA_PORTTYPE_UNKNOWN	6.5.1.3.2
PortState	M	O	SMHBA2_PORTSTATE_UNKNOWN	6.5.1.3.3
OSDeviceName	M	O	null string	6.5.1.3.4
PortSpecificAttributes	M	M	not applicable	6.5.1.3.5

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.7 FC_Port Attributes

Table A.6 specifies the requirements for implementing FC_Port Attributes for software that is compliant with this standard. Different requirements are specified for attributes of local end ports than for discovered FC_Ports.

Table A.6 — FC_Port Attributes (part 1 of 2)

Attribute Name	For local FC_Ports	For discovered FC_Ports	Value indicating unspecified	Reference
NodeWWN	M	O	eight null bytes	6.5.2.3.1
PortWWN	M	M	not applicable	6.5.2.3.2
AddressIdentifier	M	M	not applicable	6.5.2.3.3
PortSupportedClassofService	M	O	zero	6.5.2.3.4

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

Table A.6 — FC_Port Attributes (part 2 of 2)

Attribute Name	For local FC_Ports	For discovered FC_Ports	Value indicating unspecified	Reference
PortSupportedFc4Types	M	O	32 null bytes	6.5.2.3.5
PortActiveFc4Types	M	M	not applicable	6.5.2.3.6
PortSymbolicName	O	O	null string	6.5.2.3.7
NumberofDiscoveredPorts	M	N	zero	6.5.2.3.8

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.8 SAS Port Attributes

Table A.7 specifies the requirements for implementing SAS Port Attributes for software that is compliant with this standard. Different requirements are specified for attributes of local end ports than for discovered SAS Ports.

Table A.7 — SAS Port Attributes

Attribute Name	For local end ports	For discovered Ports	Value indicating unspecified	Reference
PortProtocol	M	M	not applicable	6.5.3.3.1
LocalSASAddress	M	M	not applicable	6.5.3.3.2
AttachedSASAddress	O	O	eight null bytes	6.5.3.3.3
NumberofDiscoveredPorts	M	O	zero	6.5.3.3.4

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.9 Generic Phy Attributes

Table A.8 specifies the requirements for implementing Phy Attributes for software that is compliant with this

standard.

Table A.8 — Generic Phy Attributes

Attribute Name	For Phy on a local end port	Value indicating unspecified	Reference
PhyHandle	M	not applicable	6.6.1.3.1
PhyType	M	not applicable	6.6.1.3.2
PhySpecificAttributes	M	not applicable	6.6.1.3.3

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.10 FC Phy Attributes

Table A.9 specifies the requirements for implementing FC Phy Attributes for software that is compliant with this standard.

Table A.9 — FC Phy Attributes (part 1 of 2)

Attribute Name	For FC Phy on a local end port	Value indicating unspecified	Reference
PhyOptions	O	zero	6.6.2.3.1
PhySupportedSpeed	O	SMHBA2_FCPHYSPEED_UNKNOWN	6.6.2.3.2
PhySpeed	O	SMHBA2_FCPHYSPEED_UNKNOWN	6.6.2.3.3

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

Table A.9 — FC Phy Attributes (part 2 of 2)

Attribute Name	For FC Phy on a local end port	Value indicating unspecified	Reference
PhyState	M	not applicable	6.6.2.3.4
PhyTopology	O	SMHBA2_FCTOPOLOGY_UNKNOWN	6.6.2.3.5
MediaType	M	not applicable	6.6.2.3.6
MaxFrameSize	M	not applicable	6.6.2.3.7

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.11 SAS Phy Attribute

Table A.10 specifies the requirements for implementing SAS Phy Attributes for software that is compliant with this standard.

Table A.10 — SAS Phy Attributes (part 1 of 2)

Attribute Name	For SAS Phy on a local end port	Value indicating unspecified	Reference
PhyIdentifier	M	not applicable	6.6.3.3.1
NegotiatedLinkRate	O	HBA_SASSTATE_UNKNOWN or HBA_SASSTATE_DISABLED or HBA_SASSTATE_FAILED.	6.6.3.3.2
ProgrammedMinLinkRate	M	not applicable	6.6.3.3.3

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

Table A.10 — SAS Phy Attributes (part 2 of 2)

Attribute Name	For SAS Phy on a local end port	Value indicating unspecified	Reference
HardwareMinLinkRate	M	not applicable	6.6.3.3.4
ProgrammedMaxLinkRate	M	not applicable	6.6.3.3.5
HardwareMaxLinkRate	M	not applicable	6.6.3.3.6
domainPortWWN	M	not applicable	6.6.3.3.7

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.12 Enet Phy Attributes

Table A.11 specifies the requirements for implementing Enet Phy Attributes for software that is compliant with this standard.

Table A.11 — Enet Phy Attributes

Attribute Name	For Enet Phy on a local end port	Value indicating unspecified	Reference
PhyOptions	O	zero	6.6.3.3.1
PhySupportedSpeed	O	SMHBA2_ENETSPEED_UNKNOWN	6.6.3.3.2
PhySpeed	M	SMHBA2_ENETSPEED_UNKNOWN	6.6.3.3.3
PhyState	M	not applicable	6.6.3.3.4
MediaType	M	not applicable	6.6.3.3.5
MaxFrameSize	M	not applicable	6.6.3.3.6
VLANMask	O	zero	6.6.3.3.7

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.13 Generic N_Port Controller Attributes

Table A.12 specifies the requirements for implementing generic N_Port Controller Attributes for software that is compliant with this standard.

Table A.12 — Generic N_Port Controller Attributes

Attribute Name	For N_Port Ctlr on a local end port	Value indicating unspecified	Reference
NPCHandle	M	not applicable	6.7.1.3.1
NPCType	M	not applicable	6.7.1.3.2
NPCSpecificAttributes	M	not applicable	6.7.1.3.3

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.14 FC N_Port Controller Attributes

Table A.13 specifies the requirements for implementing FC N_Port Controller Attributes for software that is compliant with this standard.

Table A.13 — FC N_Port Controller Attributes (part 1 of 2)

Attribute Name	For FC N_Port Ctlr on a local end port	Value indicating unspecified	Reference
FCNOptions	M	not applicable	6.7.2.3.1

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

Table A.13 — FC N_Port Controller Attributes (part 2 of 2)

Attribute Name	For FC N_Port Ctlr on a local end port	Value indicating unspecified	Reference
VFIDMask	O	zero	6.7.2.3.2
CoreNPortName	M	not applicable	6.7.2.3.3
CoreSwitchName	M	not applicable	6.7.2.3.4
PortVfid	O	zero	6.7.2.3.5

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.15 ENode FCoE Controller Attributes

Table A.14 specifies the requirements for implementing ENode FCoE Controller Attributes for software that is compliant with this standard.

Table A.14 — ENode FCoE Controller Attributes

Attribute Name	For ENode FCoE Ctlr on a local end port	Value indicating unspecified	Reference
FCoECtlrOptions	M	not applicable	6.7.3.4.1
ENodeMAC	M	not applicable	6.7.3.4.2
FCoECtlrVLTag	O	zero	6.7.3.4.3
DiscoveredFCFMACAddrList	O	null	6.7.3.4.4

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.16 FCoE Link Endpoint

Table A.15 specifies the requirements for implementing FCoE Link Endpoint for software that is compliant with this standard.

Table A.15 — FCoE Link Endpoint

Attribute Name	For FCoE Link Endpoint	Value indicating unspecified	Reference
LEPVNPortMAC	O	eight null bytes	6.7.4.3.1
LEPFCFMAC	O	eight null bytes	6.7.4.3.2
LEPVLTag	O	zero	6.7.4.3.3
BeaconPeriod	O	zero	6.7.4.3.4
FKAADVPeriod	O	zero	6.7.4.3.5

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.17 Fabric Info

Table A.16 specifies the requirements for implementing Fabric Info for software that is compliant with this standard.

Table A.16 — Fabric Info

Attribute Name	Requirement	Value indicating unspecified	Reference
FabricHandle	M	not applicable	6.8.1.3.1
FabricName	M	not applicable	6.8.1.3.2
Flags	O	zero	6.8.1.3.3
Ratov	O	zero	6.8.1.3.4
Edtov	O	zero	6.8.1.3.5

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.18 Protocol Statistics

Table A.17 specifies the requirements for implementing protocol statistics for software that is compliant with this standard. For any protocol statistic, the value indicating unspecified shall be negative one (-1).

Table A.17 — Protocol Statistics

Attribute Name	Requirement	Reference
SecondsSinceLastReset	O	6.9.1.3.1
InputRequests	O	6.9.1.3.2
OutputRequests	O	6.9.1.3.3
ControlRequests	O	6.9.1.3.4
InputMegabytes	O	6.9.1.3.5
OutputMegabytes	O	6.9.1.3.6

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.19 Port Statistics

Table A.18 specifies the requirements for implementing port statistics for software that is compliant with this standard. For any port statistic, the value indicating unspecified shall be negative one (-1).

Table A.18 — Port Statistics (part 1 of 2)

Attribute Name	Requirement	Reference
SecondsSinceLastReset	O	6.9.2.3.1

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

Table A.18 — Port Statistics (part 2 of 2)

Attribute Name	Requirement	Reference
TxFrames	O	6.9.2.3.2
RxFrames	O	6.9.2.3.3
TxWords	O	6.9.2.3.4
RxWords	O	6.9.2.3.5

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.20 SAS Phy Statistics

Table A.19 specifies the requirements for implementing SAS Phy statistics for software that is compliant with this standard. For any SAS Phy, the value indicating unspecified shall be negative one (-1).

Table A.19 — SAS Phy Statistics

Attribute Name	Requirement	Reference
SecondsSinceLastReset	O	6.9.4.3.1
TxFrames	O	6.9.4.3.2
TxWords	O	6.9.4.3.3
RxFrames	O	6.9.4.3.4
RxWords	O	6.9.4.3.5
InvalidDwordCount	M	6.9.4.3.6
RunningDisparityErrorCount	M	6.9.4.3.7
LossofDwordSyncCount	M	6.9.4.3.8
PhyResetProblemCount	M	6.9.4.3.9

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.21 FC Phy Statistics

Table A.20 specifies the requirements for implementing FC Phy statistics for software that is compliant with this standard. For any FC Phy statistic, the value indicating unspecified shall be negative one (-1).

Table A.20 — FC Phy Statistics

Attribute Name	Requirement	Reference
SecondsSinceLastReset	O	6.9.5.3.1
TxFrames	O	6.9.5.3.2
RxFrames	O	6.9.5.3.3
TxWords	O	6.9.5.3.4
RxWords	O	6.9.5.3.5
LIPCount	O	6.9.5.3.6
NOSCount	O	6.9.5.3.7
ErrorFrames	O	6.9.5.3.8
DumpedFrames	O	6.9.5.3.9
LinkFailureCount	M	6.9.5.3.10
LossOfSyncCount	M	6.9.5.3.11
LossOfSignalCount	M	6.9.5.3.12
PrimitiveSeqProtocolErrCount	M	6.9.5.3.13
InvalidTxWordCount	M	6.9.5.3.14
InvalidCRCCount	M	6.9.5.3.15
FLOGICount	O	6.9.5.3.16
FLOGOCount	O	6.9.5.3.17

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.22 Enet Phy Statistics

Table A.21 specifies the requirements for implementing Enet Phy statistics for software that is compliant with this standard. For any Enet Phy, the value indicating unspecified shall be negative one (-1).

Table A.21 — Enet Phy Statistics

Attribute Name	Requirement	Reference
SecondsSinceLastReset	O	6.9.6.3.1
TxENFrames	O	6.9.6.3.2
TxENBytes	O	6.9.6.3.3
RxENFrames	O	6.9.6.3.4
RxENBytes	O	6.9.6.3.5
LinkFailureCount	O	6.9.6.3.6
SymbolErrorCount	O	6.9.6.3.7
ErroredBlockCount	O	6.9.6.3.8
FCSErrorCount	O	6.9.6.3.9

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.23 FIP Statistics

Table A.22 specifies the requirements for implementing FIP Statistics for software that is compliant with

this standard. The value indicating unspecified shall be negative one (-1).

Table A.22 — FIP Statistics

Attribute Name	Requirement	Reference
SecondsSinceLastReset	O	6.9.7.3.1
FIPVLANNotifications	M	6.9.7.3.2
FCFTimeoutCount	M	6.9.7.3.3
BEACONTimeoutCount	M	6.9.7.3.4
FIPMulticastAdvertRecievedCount	M	6.9.7.3.5
KeepAlivesSentCount	M	6.9.7.3.6
ClearVirtualLinksReceivedCount	M	6.9.7.3.7

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

A.24 FC-3 Management Attributes

Table A.23 specifies the requirements for implementing FC-3 management attributes for software that is compliant with this standard. The value of each attribute shall be the value in the corresponding field in an RNID LS_ACC transmitted by the local end port.

A compliant function shall make no changes to any attribute not allowed to be set.

Table A.23 — FC-3 Management Attributes

Attribute Name	Getting [a]	Setting [a]	Reference
WWN [a]	M	N	6.13.3.2
unittype	M	O	6.13.3.3
PortId	M	O	6.13.3.4
NumberOfAttachedNodes	M	O	6.13.3.5
IPVersion	M	O	6.13.3.6
UDPPort	M	O	6.13.3.7
IPAddress	M	O	6.13.3.8
TopologyDiscoveryFlags	M	O	6.13.3.9

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

[a] The value of the WWN attribute shall be the first eight bytes of the value in the corresponding field in an RNID LS_ACC transmitted by the local end port.

A.25 SM-HBA-2 Library Attributes

Table A.24 specifies the requirements for implementing library attributes for software that is compliant with this standard. For any Library Attribute, the value indicating unspecified shall be negative one (-1).

Table A.24 — Library Attributes

Attribute Name	Requirement	Reference
LibPath	M	6.12.6.2
VName	M	6.12.6.3
VVersion	M	6.12.6.4
tm_mday	M	6.12.6.5
tm_mon	M	6.12.6.5
tm_year	M	6.12.6.5

Key:
M designates a function that is mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Note - Support requirements for mandatory, optional, and not allowed are described in A.1.

Annex B
(Informative)

Bibliography

The following are not normative but provide important background for understanding this standard. For information on the current status of the listed document(s), or regarding availability, contact the indicated organization.

SNIA HBA API: *SNIA Common HBA API* Version 2.18, March 1, 2002

NOTE 32 - The SNIA is the Storage Networking Industry Association. Information about the availability of its publications may be obtained from the SNIA by writing to 425 Market Street, Suite 1020, San Francisco, CA 94105 USA, by telephone to (USA) 415.402.0006, or on the World Wide Web at http://www.snia.org/about/contact_us/.