Working Draft

Project T11/1568-D

Revision 8
16 April 2003

Information technology - Fibre Channel HBA API (FC-HBA)

T11 Technical Editor: Robert Nixon
Emulex
3535 Harbor Blvd.
Costa Mesa, CA 92626
USA

Telephone: 714-513-8117
Facsimile: 714-513-8265
Email: bob.nixon@emulex.com

Reference number
ISO/IEC XXXXX-XXX : 200x
ANSI INCITS.*:200x**

Printed Wednesday, April 16, 2003 4:41 PM

Points of Contact:

T11 Chair	**T11 Vice-Chair**
Robert Snively	Edward Grivna
Brocade Communications Systems Incorporated	Cypress Semiconductor
1745 Technology Drive	2401 E. 86th Street
San Jose, CA 95110	Bloomington, MN 55425
Tel: 408-487-8135	Tel: 952-851-5046
Fax: 408-392-6676	Fax: 612-851-5087
Email: rsnively@brocade.com	Email: elg@cypress.com

INCITS Secretariat	Telephone:	202-737-8888
1250 Eye Street, NW Suite 200	Facsimile:	202-638-4922
Washington, DC 20005	Email:	incits@itic.org

T11 Web Site www.t11.org

T11.2 Reflector
To subscribe send e-mail to t11_2-request@mail.t11.org with ‘subscribe’ as the subject
To unsubscribe send e-mail to t11_2-request@mail.t11.org with ‘unsubscribe’ as the subject
Internet address for distribution via T11.2 reflector: t11_2@mail.t11.org

T11.3 Reflector
To subscribe send e-mail to t11_3-request@mail.t11.org with ‘subscribe’ as the subject
To unsubscribe send e-mail to t11_3-request@mail.t11.org with ‘unsubscribe’ as the subject
Internet address for distribution via T11.3 reflector: t11_3@mail.t11.org

T11.5 Reflector
To subscribe send e-mail to t11_5-request@mail.t11.org with ‘subscribe’ as the subject
To unsubscribe send e-mail to t11_5-request@mail.t11.org with ‘unsubscribe’ as the subject
Internet address for distribution via T11.5 reflector: t11_5@mail.t11.org

Document Distribution		
Global Engineering	Telephone:	303-792-2181 or
15 Inverness Way East		800-854-7179
Englewood, CO 80112-5704	Facsimile:	303-792-2192

Revision Information

1 Revision History

1.1 Revision 1 (May 7, 2002)

a) Originated from SNIA submission “Common HBA API Version 2.18” dated March 1, 2002 (ref. T11/02-149v0)
b) Adapted to the T10 Working Draft template (T10/01-314r0)
c) Modified to reflect T11 as the committee of origin
d) Extracted material from the original SNIA Introduction and from the SD3 for this project to compose the Foreword
e) Added Normative References and Definitions, Symbols, Abbreviations, and Conventions clauses

1.2 Revision 2 (May 24, 2002)

a) Corrected the document title and status
b) Corrected the revision history for Revision 1
c) Added content for clause 1, Scope
d) Added definitions and abbreviations for application programming interface, host bus adapter, storage area network, persistent binding, target mapping, wrapper library, vendor specific library, and several basic SCSI terms
e) Removed prior clause 4, which was redundant with the Foreword
f) Added a new clause 4 that specifies general design constraints on the specification and implementations.
g) Merged prior clauses 5, 11, 12, and 13 to create a new clause 5 specifying software structure and behavior rules
h) Removed prior clause 6, which was not relevant to standard specification
i) Created a new clause 6 by merging the prior clauses 7 (Attribute Definitions) and 8 (Data Structures)
j) Prior clause 9 (Function Calls) becomes new clause 7
k) Prior clause 10 becomes new clause 8
l) Extracted descriptive material on target mapping and persistent binding to create Annex A
m) Extracted all coding samples to create Annex B

1.3 Revision 3 (July 29, 2002)

a) Corrected typo (“APIVersion”) and spelling error (“Foreward”) in revision history for Revision 1
b) Added function HBA_SendRLS per 02-299v0
c) Removed Port type definitions for E_Ports and G_Ports per meeting 6/11/02
d) Clarified relationship of HBA_MgmtInfo and RNID per meeting 6/11/02
e) Removed comments from data structure definitions where similar textual field descriptions closely follow.
f) Changed many places to make consistent use of glossary terminology per meeting 6/11/02.
g) Changed many places to make consistent use of normative keywords per meeting 6/11/02.
h) Clarified memory layout of certain data structures containing IP Addresses, SCSI LUNs, Name_Identifiers, and address identifiers per meeting 6/11/02.
i) Clarified restrictions on values of certain data types specified as lists of symbolic constants, per meeting 6/11/02.: See specifications of data types HBA_STATUS, HBA_PORTTYPE, HBA_PORTSTATE, HBA_PORTSPEED, HBA_BIND_CAPABILITY, HBA_BIND_TYPE, EventCode, and EventType.
j) Eliminated trademarked names from examples in clause 8
k) Added function directory and requirements table per meeting 6/11/02
l) Added overview descriptions of polled and asynchronous event reporting methods per meeting 6/11/02. Some of the material used was removed from the description of function HBA_GetEventBuffer.

1.4 Revision 4 (August 30, 2002)

All numeric references are with respect to the enumeration of version 3

a) Fixed typos
   A) In 3.1.7 changed “TA” to “A”;
   B) In 6.4.2.8 and 6.4.2.9 header lines, changed “4ypes” to “4Types”;
   C) In 7.3.10.2, changed “bet” to “be”.
   D) In 7.4.4.4, changed “SAM-2SAM-2” to “SAM-3”.
   E) In 7.5, changed several “HBA_STATUS_CHECK_CONDITION“ to “HBA_STATUS_SCSI_CHECK_CONDITION“.
   F) In 7.6.6.1 change “,,” to “,”.
   G) In 7.6.8.3 description of object_wwn, change “object_wwn is nonzero” to “object_wwn is zero”.
   H) In A.4.1 change “to to” to “to”.
   I) In A.4.8, change “the the” to “the”.
b) In 6.10.1, added entries for HBASendRLS to the list of function prototype declarations and to the declaration for structure HBA_EntryPointsV2. Necessary to complete 02-299v0.
c) In 6.4.2.14, clarified the scope of the NumberOfDiscoveredPorts reported for a local Nx_Port. See 02-305v1.
d) In 7.5, specified in the description of each function that it shall not cause a SCSI command to be sent to an Nx_Port that is not a SCSI target, and specified in the returned function value for each function that HBA_STATUS_ERROR_NOT_A_TARGET shall be returned if the remote Nx_Port identified in the function call does not have SCSI Target functionality. See 02-305v1.
e) In 7.4.4.2 and 7.4.5.2, clarified that retrieved target mapping lists should be current at the time of the function call that retrieves them. See 02-305v1.
f) In 5.4.2 and 7.2.9.2, changed the use of HBA_STATUS_ERROR_STALE_DATA and its related call to HBA_RefreshInformation from optional to mandatory. See 02-305v1and 02-495v0.
g) In 7.4.4.4, clarified that HBA_STATUS_ERROR_MORE_DATA is a valid returned function value for HBA_GetFcpTargetMapping. See 02-305v1.
h) In 7.4.6.4, clarified that HBA_STATUS_ERROR_MORE_DATA is a valid returned function value for HBA_GetFcpPersistentBinding. See 02-305v1.
i) In 6.2, added a function return code for aborting a function that would have caused a SCSI overlapped command condition. In 7.5, specified in the description of each function that it shall not cause a SCSI command to be sent if it would cause a SCSI overlapped command condition, and specified in the returned function code for each function that HBA_STATUS_ERROR_TARGET_BUSY shall be returned if the implementation is unable to send the requested command without causing a SCSI overlapped command condition. See 02-305v1and 02-495v0.
j) In 3.1.63, 7.4.4.2, and 7.4.5.2, clarified that a target mapping may identify either a specific logical unit or a whole target. In 6.6.2.7, 6.6.2.8, 6.6.2.10, 6.6.2.12, 6.6.2.13, and 6.6.2.14, added requirements for return values within returned target mappings. Changed all SAM-2 references to SAM-3. See 02-305v1and 02-495v0.
k) In 6.4.2.13 and 6.6.2.11, added rules and tables specifying the names to be returned as OSDeviceName. See 02-439v0 and 02-495v0.
l) In 6.2 and 7.x.x.4, clarified the constraints this standard places on the values returned as function status. See 02-495v0 concerning 02-289v1.
m) In table 1, removed the compliance information. See 02-495v0 concerning 02-407v0.
n) In 6.5.2.10, 6.5.2.11, 6.5.2.12, 6.5.2.13, 6.5.2.14, and 6.5.2.15, emphasized the requirement from 7.3.9.2 that certain statistics values shall be equal to Link Error Status Block fields.
o) In 6.9.1.2, 6.9.1.3, 6.9.1.4, 6.9.1.5, 6.9.1.6, 6.9.1.7, 7.7.3.1, 7.7.4.2, 7.7.5.2, 7.7.6.2, 7.7.7.2, 7.7.8.2, and 7.7.9.2, used the term “category” instead of the inconsistently used and inappropriately suggestive term “level” to label groupings of event types. In the glossary, added “event”, “event category”, and “event type”.
p) In 6.9.2.1, added descriptions of the asynchronous event types.
q) In many places, substituted the more specific term HBA for adapter.

1.5 Revision 5 (25 October 2002)

All numeric references are with respect to the enumeration of version 4.

a) Throughout the document, corrected many spelling problems identified by FrameMaker Spelling Checker (editorial).
b) In 1, inserted project numbers for recently authorized INCITS standards development projects (editorial).
c) In 2.3, updated the revision numbers of draft standards (editorial).
d) In 2.3, added a reference to FC-PI-2 (editorial).
e) In 3.1.10, removed spurious sentence concerning construction of unique identifiers (editorial).
f) In 3.1.21, corrected the reference documents for the definition of FC_Port (editorial).
g) In 6.2, add a definition for currently undefined status code HBA_STATUS_ERROR_ILLEGAL_FCID (editorial).
h) In 7.3.5.1, change pAdapterattributes to indicate that it is a pointer (editorial).
i) In 6.6.1.7, remove a spurious space from the middle of the reference to structure HBA_BIND_TYPE in the declaration of HBA_FCPBindingEntry2 (editorial).
j) In 6.6.1.7, add a semicolon after the Status field in the declaration of HBA_FCPBindingEntry2 (editorial).
k) In 7.4.5.1, remove the semicolon after the definition of the pMapping field (editorial).
l) In 7.4.9.1, add a space between the words in the definition of the binding field (editorial).
m) In 7.4.10.1, add a comma after the definition of the handle field and remove the comma after the definition of the hbaPortWWN field (editorial).
n) In 7.4.11.1, remove extra tabs before the definition of the statistics field (editorial).
o) In many places, incorporated FC-HBA compliance requirements (per 02-407v3, approved FC-HBA meeting 10/8/02)
p) In references, glossary, and new annex B, incorporated HBA API Mapping to InfiniBand (per 02-425v2, approved FC-HBA meeting 10/8/02)

1.6 Revision 6 (19 December 2002)

All numeric references are with respect to the enumerations of version 5.

a) Corrected several spelling problems identified by FrameMaker Spelling Checker (editorial).
b) Repunctuated many lists to make them more nearly compliant with the T10 editors style guide. (editorial)
c) Replaced several inappropriate uses of “which”, “can”, “must”, “could”, “will”, “assume”, “execute”, and “NCITS” with appropriate alternatives (editorial)
d) Replaced several possibly ambiguous references to “them” by more specific references. (editorial)
e) Replaced all instances of Name Identifier with Name_Identifier for consistency with FC_FS and FC-GS-4. (editorial)
f) Resolved all editor's notes concerning units for sizes to specify sizes are in bytes. (per 02-550v1, approved at FC-HBA meeting December 10, 2002)
g) In 2.2 and 2.3, updated the version numbers of references. (editorial)
h) In 2.2, 2.3, 3.1, 3.2, 6, and 7, added specifications for support of SB devices. (per 02-423v4, approved at FC-HBA meeting December 10, 2002)
i) In 3.1, 4.1, 4.2, 5.2, 5.3, 5.5, several places in 7, 8.1, and several places in Annex A, establish additional grounds of compliance for products that have the required external interface but that achieve multivendor configurability by OS-supported structures. (per 02-550v1, approved at FC-HBA meeting December 10, 2002)
j) In 6.2 and 7.2.2.4, added an error status code to indicate that a call has been made to HBA_LoadLibrary when the library is already loaded. (per 02-550v1, approved at FC-HBA meeting December 10, 2002)
k) In 6.2 and 7.2.3.4, added error status codes to indicate that a call has been made to HBA_FreeLibrary when there is no library loaded. (per 02-550v1, approved at FC-HBA meeting December 10, 2002)

l) In 6.4.2.14 table 1, revise the preferred format for the name of an HBA local adapter port on Solaris to reflect the latest recommendation from SNIA (per 02-550v1, approved at FC-HBA meeting December 10, 2002)
m) In 6.6.2.11 table 2, annotate the preferred formats for the names of magnetic and optical logical units on Solaris to reflect the latest recommendation from SNIA (per 02-550v1, approved at FC-HBA meeting December 10, 2002)
n) In 6.7.1, inserted missing open bracket at beginning of `HBA_MgmtInfo` structure declaration. (editorial)
o) In 7.7.3.1, resolve editor's note 20 concerning multiple registrations for the same asynchronous event from the same application by specifying the later registration becomes effective in addition to the earlier one(s). (per 02-550v1, approved at FC-HBA meeting December 10, 2002)
p) In 7.6.6.2, modify the rules for validating an HBA_SendRNIDV2 request to allow for an RNID reply that does not contain the Common Node Identification Data. (per 02-550v1, approved at FC-HBA meeting December 10, 2002)
q) In B.2.5.1, removed a paragraph that incorrectly described the DATA-OUT BUFFMT field, and that would be redundant with an earlier paragraph if corrected. (editorial)

1.7 Revision 7 (19 February 2003)

a) In the T11 cover pages, added subscription information for the T11.5 reflector (editorial).
b) In the Foreword, corrected list item b to be two list items. (editorial).
c) In the Foreword, corrected the description of Task Group T11.5 (editorial).
d) In the Foreword, changed format of membership lists to be similar to FC-FS (editorial).
e) In the Foreword, removed redundant clause/annex counts from Introduction (direction of FC-HBA meeting 6 Feb 03).
f) In the Scope, made formatting corrections to first list (direction of FC-HBA meeting 6 Feb 03).
g) In the Scope, replaced entire list of “examples of standards” in Scope with reference to T10 and T11 web sites (direction of FC-HBA meeting 6 Feb 03).
h) In the Normative references, changed the fax number for ANSI customer service to match the one on their web site (editorial).
i) In the Approved references, removed uncited reference to IEEE802 (editorial).
j) In the References under development, removed revision numbers from references (direction of FC-HBA meeting 6 Feb 03).
k) In the References under development, removed uncited reference to FC-PI (direction of FC-HBA meeting 6 Feb 03).
l) In the Normative references, changed SRP reference from a reference under development to an approved reference (direction of FC-HBA meeting 6 Feb 03).
m) In the IETF references, removed uncited references to RFCs 854, 2373, 2616, 2625, and 2818 (editorial).
n) In the Normative references, removed the Other references subclause and made it an informative annex (direction of FC-HBA meeting 6 Feb 03).
o) In the Glossary, defined “local Nx_Port” (direction of FC-HBA meeting 6 Feb 03).
p) In the Glossary, clarified the definitions of N_Port and Nx_Port, including adding new definitions for Public NL_Port, Private NL_Port, and end port, and changing local Nx_Port to local end port (editorial).
q) In the Glossary, added definition of FCP_Port (editorial).
r) In Conventions, allowed only upper case for alphabetic hex digits, and allowed separating hex numbers into groups of four digits for readability (editorial).
s) In the End port statistics attribute specifications, consolidated the description of counter behavior from scattered places (editorial).
t) In the Library Attributes, provided missing function entry point declarations and function table declaration for the SB functions (edtiorial).
u) In the specification of function HBA_GetFC4Statistics, corrected a discrepancy in the name of the Port WWN parameter (editorial).
v) In the specification of function HBA_RefreshInformation, clarified the circumstance in which HBA_STATUS_ERROR_STALE_DATA shall be returned (editorial).

w) In the Compliance annex, incorporated management interoperability requirements per 03-007v1 (direction of FC-HBA meeting 6 Feb 03).
x) In the InfiniBand annex, corrected the title of the second figure (editorial).
y) In the InfiniBand annex, incorporated buffer overflow and binding query features per 03-035v1 (direction of FC-HBA meeting 6 Feb 03).
z) globally, checked spelling of InfiniBand, and added a trademark acknowledgement to critical references (direction of FC-HBA meeting 6 Feb 03).
aa) Globally, replaced “vendor specific library” with “HBA specific library” and “HBA vendor specific software” with “HBA specific software” (direction of FC-HBA meeting 6 Feb 03).
ab) Globally, corrected the spelling of N_Port_ID, N_Port_Name, and Node_Name (direction of FC-HBA meeting 6 Feb 03).
ac) Globally, replaced references to Nx_Port and N_Port with end port as appropriate (editorial).
ad) Globally, normalized the use of “i.e.” and “e.g.” (editorial).
ae) Globally, normalized appearance of parenthetic crossreferences (editorial).

1.8 Revision 8 (16 April 2003)

a) Globally, change “Fibre Channel HBA” to “HBA” (direction of FC-HBA meeting 10 April 2003).
b) Globally, change “FCP-2 SCSI” to “FCP-2” (direction of FC-HBA meeting 10 April 2003).
c) Globally, removed unnecessary quotation marks (direction of FC-HBA meeting 10 April 2003).
d) In Abstract, change reference to this standard to say “This standard” (direction of FC-HBA meeting 10 April 2003).
e) In Abstract, Foreword, and Scope, change “exercise” to “use” (direction of FC-HBA meeting 10 April 2003).
f) In the Table of Contents, adjusted all subclause levels to use the same indent (direction of FC-HBA meeting 10 April 2003).
g) In Foreword, changed “is a piece of hardware” to “is hardware” (direction of FC-HBA meeting 10 April 2003).
h) In Foreword, changed a long “that is” clause to a separate sentence (direction of FC-HBA meeting 10 April 2003).
i) In Foreword, capitalized network in Storage networking Industry Association (editorial).
j) In Foreword, corrected description of FC-MI to be an INCITS Technical Report (direction of FC-HBA meeting 10 April 2003).
k) In Foreword, removed uninformative adverbs “widely”, “generally”, and “largely” (direction of FC-HBA meeting 10 April 2003).
l) In Foreword paragraph 5, changed “Specifically, it” to “This standard” (direction of FC-HBA meeting 10 April 2003).
m) In Foreword paragraph 5, added a colon to introduce the list (direction of FC-HBA meeting 10 April 2003).
n) In Foreword list item c, changed “sufficient to satisfy“ to “necessary to comply with“ (direction of FC-HBA meeting 10 April 2003).
o) In Foreword list item f, added “, and“ before the semicolon at the end (direction of FC-HBA meeting 10 April 2003).
p) In Introduction, change “The Fibre Channel HBA API (FC-HBA) standard“ to “This standard“ (direction of FC-HBA meeting 10 April 2003).
q) Removed the Acknowledgements (direction of FC-HBA meeting 10 April 2003).
r) In Scope paragraph 1, changed the first “This standard” to “The Fibre Channel HBA API standard” (direction of FC-HBA meeting 10 April 2003).
s) In Scope paragraph 1, changed “Specifically, this standard” to “This standard” (direction of FC-HBA meeting 10 April 2003).
t) In Scope paragraph 1, added a colon to introduce the list (direction of FC-HBA meeting 10 April 2003).
u) In Scope list item c, changed “sufficient to satisfy“ to “necessary to comply with“ (direction of FC-HBA meeting 10 April 2003).
v) In Scope last paragraph, remove “examples of “ (direction of FC-HBA meeting 10 April 2003).
w) In 2.3, added a reference to FC-SW-3 (direction of FC-HBA meeting 10 April 2003).

x) In 3.1, added the acronym to all terms in the glossary for which the acronym is in 3.2 (direction of FC-HBA meeting 10 April 2003).
y) In 3.1.1, removed a spurious colon at the end of the paragraph title (direction of FC-HBA meeting 10 April 2003).
z) In 3.1.5, added a period at the end of the paragraph (direction of FC-HBA meeting 10 April 2003).
aa) In 3.1.14, added an example of concatenation (direction of FC-HBA meeting 10 April 2003).
ab) In 3.1.18, corrected the format of the hexadecimal number (direction of FC-HBA meeting 10 April 2003).
ac) In 3.1.22, removed redundant references to standards FC-FS and FC-PI (direction of FC-HBA meeting 10 April 2003).
ad) In 3.1.24, changed reference standards to FC-AL-2 and FC-SW-3 (direction of FC-HBA meeting 10 April 2003).
ae) In 3.1.27, removed redundant reference to standards FC-GS-4 (direction of FC-HBA meeting 10 April 2003).
af) In 3.1.37, changed the definition of Internet Protocol to match the better one from FC-SWAPI (direction of FC-HBA meeting 10 April 2003).
ag) In 3.1.39, corrected errors in the definition of Loop Initialization Primitive (direction of FC-HBA meeting 10 April 2003).
ah) In 3.1.43, corrected errors in the definition of FL_Port (direction of FC-HBA meeting 10 April 2003).
ai) In 3.1.47, corrected the name of the Not_Operational Primitive Sequence (direction of FC-HBA meeting 10 April 2003).
aj) In 3.1.48, replaced the undefined acronym LCF with Link Control Facility (direction of FC-HBA meeting 10 April 2003).
ak) In 3.1.64, qualified OPN as a primitive signal (direction of FC-HBA meeting 10 April 2003).
al) In definitions, added a definition of FCP-2 (direction of FC-HBA meeting 10 April 2003).
am) In definitions, added a definition of HBA API instance (direction of FC-HBA meeting 10 April 2003).
an) In 3.1.31, made the definition of host bus adapter specific to Fibre Channel (direction of FC-HBA meeting 10 April 2003).
ao) In Annex B.2.6.1, allow vendor specific binding establishment (03-206v0, approved at the FC-HBA meeting 10 April 2003)
ap) In Annex B.2.7.1, allow selective return of binding information (03-206v0, approved at the FC-HBA meeting 10 April 2003)
aq) In Annex C, corrected the title from “LUN Mapping ...” to “Target Mapping ...”

ANSI (r)
INCITS.*:200x**

American National Standards for Information Systems -

Fibre Channel HBA API (FC-HBA)

Secretariat
InterNational Committee for Information Technology Standards

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 Host Bus Adapters and use of certain Fibre Channel facilities for discovery and management of the components of a Fibre Channel Storage Area Network.

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

Published by
American National Standards Institute
11 West 42nd Street, New York, NY 10036

Contents

List of Tables

List of Figures

Foreword

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

An HBA is hardware, typically on a host system and sometimes embedded on a RAID controller or other storage device that interfaces a system to a Fibre Channel fabric. An HBA may support multiple FC-4 types, including FCP-2, IPFC, and FC-VI. In order to manage an HBA and the fabric behind it, upper level management software applications require information that has not been available from HBAs in a consistent manner across operating systems, vendors, and platforms, and in some cases not at all. Implementations have been HBA vendor specific. This required that when a software application needed access to certain Fibre Channel parameters (e.g.. WWN or attached LUNs), vendor specific drivers or OS specific calls had to be used to get to this information. This has resulted in long qualification times, difficult integration across platforms, and inconsistency between HBA vendors. It further has made implementation of cross-vendor SAN management applications difficult and expensive in development time and code complexity. A standard HBA API enables implementation by multiple vendors of a consistent low-level HBA interface for accessing information in a Fibre Channel Storage Area Network.

An initial specification for a common HBA API (HBA API Phase 1) was prepared within The Storage Networking Industry Association (The SNIA) during 2000 and published as an informative annex to INCITS Technical Report FC-MI. It has been adopted by vendors of SAN management software and HBAs, and is recognized as having resolved the perception of Fibre Channel as being unmanageable. It has since become the basis for definition of new FC-GS-4 Fabric Services as well.

SNIA has provided a proposal for the initial draft of the HBA API Phase 2 that resolves many omissions and issues with the HBA API Phase 1 and is fully compatible with the earlier specification. SNIA HBA API Phase 2 is the basis for this standard.

This standard defines an Application Programming Interface the scope of which is management of Fibre Channel Host Bus Adapters and use of certain Fibre Channel facilities for discovery and management of the components of a Fibre Channel Storage Area Network. This standard defines interfaces to capabilities for:

a) observation and modification of descriptive and operational characteristics of HBAs and end ports;
b) access to Fibre Channel Fabric Services;
c) access to Fibre Channel Extended Link Services necessary to comply with the FC-MI manageability profile for Host Bus Adapters;
d) discovery and characterization of FCP-2 storage resources;
e) observation of HBA, end port, and storage access traffic statistics;
f) observation and modification of the availability and representation of Fibre Channel storage resources to Operating System applications; and
g) timely and selective reporting of HBA and fabric configuration, status, and statistical events.

With any technical document there may arise questions of interpretation as new products are implemented. INCITS has established procedures to issue technical opinions concerning the standards developed by INCITS. These procedures may result in Technical Information Bulletins being published by INCITS.

These Bulletins, while reflecting the opinion of the Technical Committee that developed the standard, are intended solely as supplementary information to other users of the standard. This standard, ANSI INCITS.***:200x, as approved through the publication and voting procedures of the American National Standards Institute, is not altered by these bulletins. Any subsequent revision to this standard may or may not reflect the contents of these Technical Information Bulletins.

Current INCITS practice is to make Technical Information Bulletins available through:

Global Engineering	Telephone:	303-792-2181 or
15 Inverness Way East		800-854-7179
Englewood, CO 80112-5704	Facsimile:	303-792-2192

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.

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

Editors Note 1 - xxx: <<Insert INCITS member list>>

Editors Note 2 - xxx: insert current T11 membership after all T11 comment resolution is complete

Editors Note 3 - xxx: insert current T11.5 membership after all T11 comment resolution is complete

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 normative specification of the mapping of HBA API functions onto InfiniBand.
Annex C is informative material that describes the motivation for target mapping and persistent binding features of HBAs.
Annex D is informative material that provides exemplary C code for several of the functions specified in the body of this standard.
Annex E is a bibliography of documents that are informatively referenced by this standard.

American National Standard **INCITS.***:200x**

American National Standard for Information Systems - Information Technology - Fibre Channel HBA API (FC-HBA)

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. The Fibre Channel HBA API standard specifies a standard API the scope of which is management of Fibre Channel host bus adapters (HBAs) and use of certain Fibre Channel facilities for discovery and management of the components of a Fibre Channel storage area network (SAN). This standard defines interfaces to capabilities for:

a) observation and modification of descriptive and operational characteristics of HBAs and end ports;
b) access to Fibre Channel Fabric Services (see FC-GS-4);
c) access to Fibre Channel Extended Link Services necessary to comply with the manageability profile for HBAs recommended in FC-MI (see FC-MI);
d) discovery and characterization of FCP-2 storage resources (see FCP-2);
e) observation of HBA, end port, and storage access traffic statistics;
f) observation and modification of the availability and representation of Fibre Channel storage resources to operating system applications; and
g) timely and selective reporting of HBA and fabric configuration, status, and statistical events.

This standard is to be used in conjunction with the Fibre Channel and SCSI families of standards, to which at the time this standard was written it related as indicated in figure 1

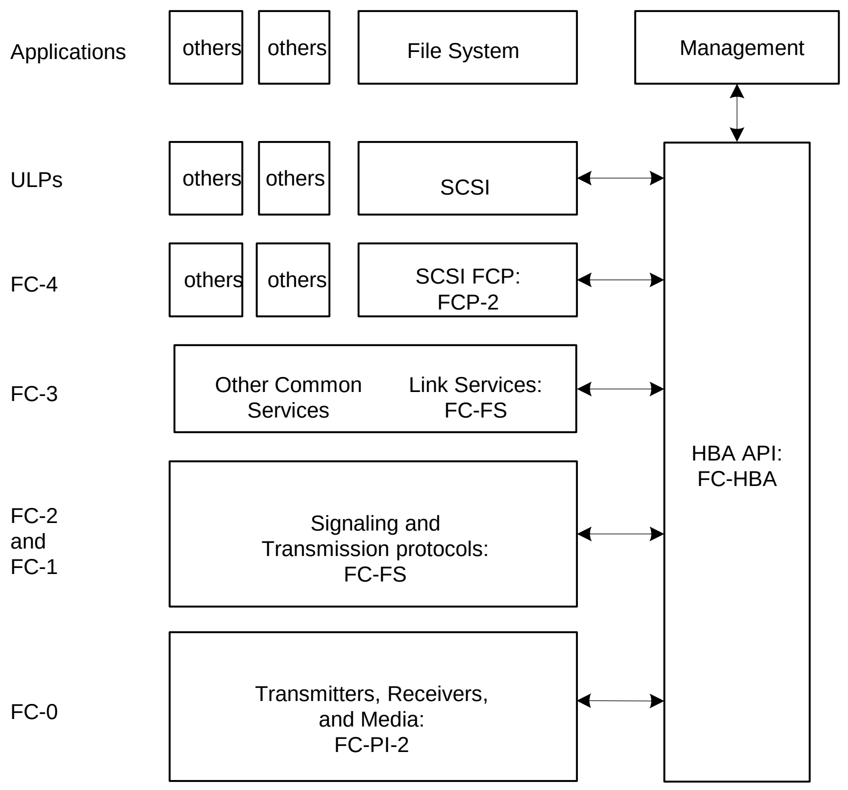


Figure 1 — Context for FC-HBA

For standards in the Fibre Channel 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 indicated 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.

Unless noted otherwise, copies of the following documents may be obtained from ANSI: approved ANSI standards, approved and draft international and regional standards (ISO, IEC, CEN/CENELEC, ITUT), and approved and draft foreign standards (including BSI, JIS, and DIN). For further information, contact ANSI Customer Service Department at 212-642-4900 (phone), 212-398-0023 (fax) or via the World Wide Web at http://www.ansi.org.

Additional availability contact information is provided below as needed.

2.2 Approved references

FC-AL-2: ANSI INCITS 332-1999, Fibre Channel – Arbitrated Loop – 2

FCP-2: INCITS 350 DRAFT, Fibre Channel Protocol-2

FC-FLA: ANSI INCITS TR-20:1998 Fibre Channel – Fabric Loop Attach

FC-SB-2: ANSI INCITS 349-2001, Fibre Channel – Single Byte Command Set – 2

SBCON: ANSI INCITS 296:1997, Single Byte Command Code Sets Connection

SRP: ANSI INCITS 365-2002 , SCSI RDMA Protocol (SRP)

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 indicated.

FC-FS: ANSI INCITS Project 1331-D, Fibre Channel - Framing and Signaling Interface

FC-GS-4: ANSI INCITS Project 1505-D, Fibre Channel-Generic Services - 4

FC-PI-2: ANSI INCITS Project 1506-D, Fibre Channel-Physical Interfaces - 2

FC-SB-3: ANSI INCITS Project 1569-D, Fibre Channel – Single Byte Command Set – 3

FC-SW-3: ANSI INCITS Project 1508-D, Fibre Channel – Switch Fabric – 3

SAM-3: ANSI INCITS Project 1561-D, SCSI Architecture Model - 3

SPC-3: ANSI INCITS Project 1416-D, SCSI Primary Commands-3

SBC-2: ANSI INCITS Project 1417-D, SCSI Block Commands - 2

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, August1980.

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 InfiniBand Trade Organization references

Copies of reference IBA may be obtained from the InfiniBand^{tm} Trade Organization, www.infinibandta.org.

IBA: InfiniBand^{tm} Architecture Specification, Volume 1, release 1.0.a.

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).

3.1.2 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. 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.

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 7.9.3.1).

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).

3.1.9 Common Transport (CT): A protocol defined by FC-GS-4 that provides access to Services and their related Servers. CT may also refer to an instance of the Common Transport (see FC-GS-4).

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: An FC-4 Device_Data frame, an FC-4 Video_Data frame, or a Link_Data frame (see FC-FS).

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).

3.1.13 Directory: A repository of information about objects that may be accessed via the Directory Service (see FC-GS-4).

3.1.14 end port: An Nx_Port or a Private NL_Port.

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

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

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

3.1.18 F_Port: The LCF 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).

3.1.19 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).

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

3.1.21 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).

3.1.22 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 and FC-PI-2).

3.1.23 FCP_Port: An end port that supports the SCSI Fibre Channel Protocol (see FCP-2).

3.1.24 FCP-2: A SCSI transport protocol for using Fibre Channel as a SCSI service delivery system (see FCP-2).

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-3).

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

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

3.1.28 Generic Services: The collection of Services defined by FC-GS-4.

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 a Fibre Channel fabric, loop, or link.

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: Two unidirectional fibres transmitting in opposite directions and their 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-3).

3.1.39 logical unit number (LUN): An encoded 64-bit identifier for a logical unit (see SAM-3).

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-3) with further constraints specified in 6.6.1.4

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: A port that contains Arbitrated Loop functions associated with Arbitrated Loop topology (see FC-AL-2).

3.1.43 Name Server: A server among those provided by Generic Services (see FC-GS-4).

3.1.44 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).

3.1.45 NL_Port: An N_Port that contains the Loop Port State Machine defined in 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.

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

3.1.47 Node_Name: A Name_Identifier associated with a node (see FC-FS).

3.1.48 Node Symbolic Name: A Symbolic Name associated with a node (see FC-GS-4).

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

3.1.50 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 and FC-AL-2).

3.1.51 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).

3.1.52 N_Port_Name: A Name_Identifier associated with an N_Port (see FC-FS).

3.1.53 Nx_Port: A 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 and FC-AL-2).

3.1.54 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

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

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

3.1.57 persistent binding: A function of an HBA that retains a pairing of an OS SCSI identification and an FCP-2 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.

3.1.58 Phase I: The Common HBA API Interface specification in FC-MI.

3.1.59 Phase II: This standard.

3.1.60 Platform: An association of one or more Nodes for the purpose of discovery and management (see FC-GS-4).

3.1.61 Port Name: A Name_Identifier associated with an FC_Port (see FC-FS).

3.1.62 Port Symbolic Name: A Symbolic Name associated with an FC_Port (see FC-GS-4).

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

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

3.1.65 Printable ASCII Characters: ASCII characters in the range 20h through 7Eh.

3.1.66 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).

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

3.1.68 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-3).

3.1.69 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-3).

3.1.70 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-4).

3.1.71 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. Examples of a service include the Directory Service and the Alias Service. A service provides access to one or more Servers (see FC-GS-4).

3.1.72 Single Byte Command Sets (SB): A Fibre Channel FC-4 protocol specified in FC-SB-2 or subsequent revisions of that standard (see FC-SB-2 or FC-SB-3).

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

3.1.74 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.

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

3.1.76 target: Synonymous with SCSI target port

3.1.77 target mapping: A function of an HBA that makes an OS SCSI identification of a target or logical unit operationally equivalent to a specified FCP-2 identification of a target or logical unit; or else a representation of a single such equivalence.

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

3.1.79 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).

3.1.80 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).

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

3.1.82 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).

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

3.1.84 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).

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

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

3.1.87 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

±	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
CM:REQ	Connection Management Request (see IBA)
FC	Fibre Channel
HBA	host bus adapter

IB	InfiniBand^{tm} Transport (see IBA)
IOC	I/O Controller (see IBA)
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
NOS	Not_Operational Primitive Sequence
OS	operating system
SCSI	Small Computer System Interface (see SAM-3)
SA	Subnet Administrator (see IBA)
SAM-3	SCSI Architecture Model-3 (see SAM-3)
SAN	storage area network
SB	Single Byte Command Code Sets (see FC-SB-2 or FC-SB-3)
SPC-3	SCSI Primary Commands-3 (see SPC-3)
SRP	SCSI RDMA Protocol (see SRP)
TCA	Target Channel Adapter (see IBA)
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

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 opaque: A keyword indicating that value has no semantics or internal structure.

3.3.8 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.

3.3.9 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.

3.3.10 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.

3.3.11 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 comma (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 shall facilitate common management methodologies for configurations of 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 fully documented by the OS vendor or as defined in this standard.

Should a fully 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).

NOTE 1 An HBA API library that is structured as a wrapper library and HBA specific libraries also allows incremental upgrade from implementations of the FC-MI Common HBA API (see 4.4).

The relationships of the components of either software structure to one another and to their environment is indicated in figure 2. Also indicated in figure 2 is that both the API offered by the HBA API 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 may be indicated elsewhere in this standard.

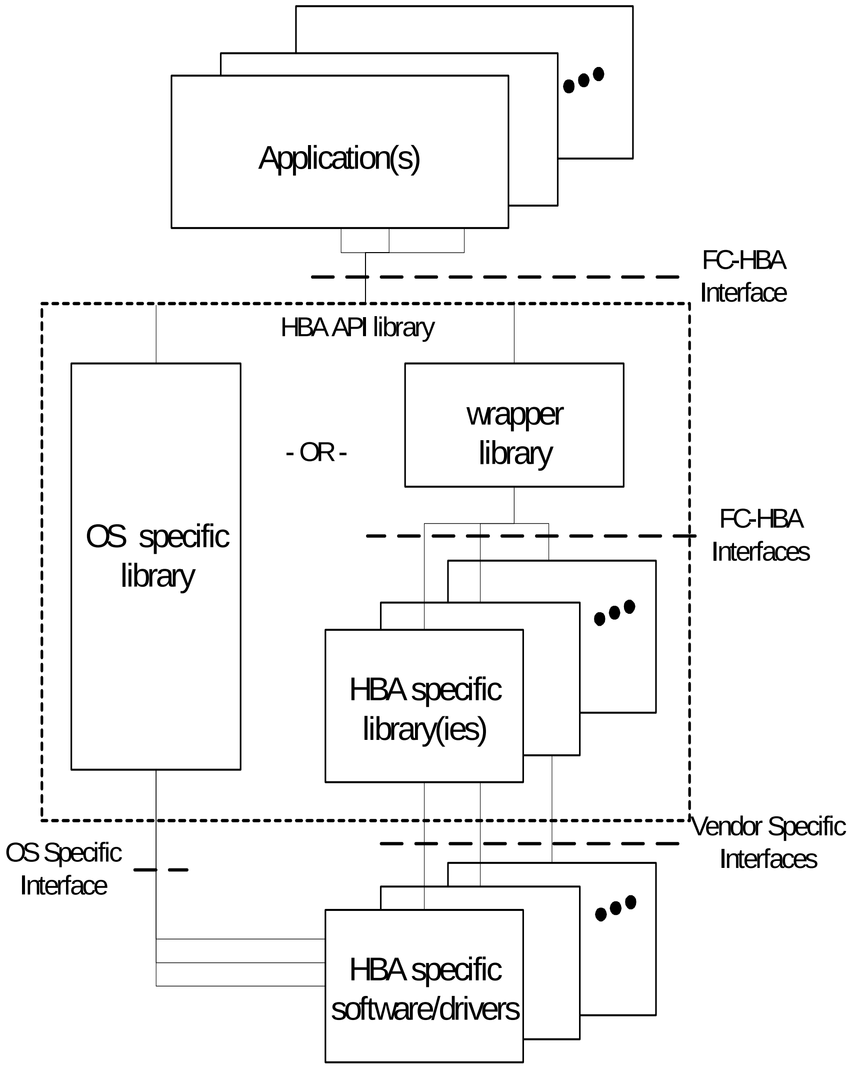


Figure 2 — Software Structure

An HBA API library that is compliant with the requirements of this specification may be identified as an FC-HBA compliant HBA API. A wrapper library that is compliant with this specification may be referenced as an FC-HBA compliant wrapper library but it shall not be referenced as an FC-HBA compliant HBA API. An HBA specific library that is compliant with this specification may be referenced as an FC-HBA compliant HBA specific library but it shall not be referenced as an FC-HBA compliant HBA API.

4.2 C language

This standard defines an API only in the C language. It is possible to specify functionally equivalent APIs in other languages but these shall not be referenced as FC-HBA 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. No constraint is specified or implied on the internal implementation of components of an HBA API library.

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

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.

4.3 Operating System Dependencies

Although most of 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 shall be understood to lead to less assurance of interoperability of compliant components in OSs for which specific normative statements have not been made.

4.4 FC-MI Common HBA API

A less featured version of the specification herein was published as an informative annex to Fibre Channel - Methodologies for Interconnect Technical Report (see FC-MI). Wrapper libraries that are compliant with this standard shall interoperate with vendor specific libraries, applications, and drivers that are compliant with the Common HBA API in FC-MI. HBA specific libraries that are compliant with this standard shall interoperate with wrapper libraries, applications, and drivers that are compliant with the Common HBA API in FC-MI. However, combinations including wrapper libraries or vendor specific libraries that are compliant only with FC-MI shall not be considered FC-HBA compliant HBA APIs.

5 Software Structure and Behavior

5.1 Overview

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

The intention of this standard is to facilitate implementation of a uniform and unitary interface to HBAs produced by multiple vendors that may be installed in the same system. The software that implements this interface may derive from components developed independently by those vendors and possibly others as well (e.g., application vendors). This standard therefore 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 further 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 standard therefore 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 those 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.
h) Any necessary initialization beyond that inherent in library loading 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 distinct 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 indicated in figure 2. For the most part, there is a one to one correspondence between the functions of the wrapper library and the functions of the HBA specific libraries. The differences are 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); however, the unitary interface goal may be compromised unless there is only a single instance of the wrapper library.

Initialization of libraries shall be implemented in the function HBA_LoadLibrary. 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 its own initialization.

References to the functions of the HBA specific libraries shall be loaded into data structures owned by the wrapper library. This shall be accomplished through the functions HBA_RegisterLibrary and HBA_RegisterLibraryV2 that shall be defined in all HBA specific libraries.

5.3 Names, Handles and Their Usage

The concepts of names and handles are used in this API as generic ways to reference an HBA. The use of a handle is specified to be independent of the operating system.

Associated with each HBA that may be opened and managed there shall be a name that:

a) shall be unique to that HBA among all applications sharing the same instance of the HBA API library; and
b) may change across OS reboots.

Associated with each HBA that may be opened and managed there shall be a handle that:

a) shall be persistent between open and subsequent close of the HBA;
b) in an HBA API library with OS independent structure its lower 16 bits shall be determined by the HBA specific library for the HBA so no two HBAs supported by the same HBA specific library have the same value;
c) in an HBA API library with OS independent structure its upper 16 bits shall be determined by the wrapper library so no two HBA specific libraries registered with that wrapper library have the same value;
d) may change across reboots; and
e) may change on successive opens of the same HBA.

These situations may be recognized in vendor specific manner:

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

When a new HBA is added, a new name shall be assigned to the HBA that does not conflict with previous names.

When an HBA is removed, the name of the HBA should not be re-used, in order for upper level software to reliably reference the same device.

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

5.4 HBA Configuration Rediscovery Effect on the API

5.4.1 Introductory discussion

The HBA API specified by this standard has several functions in which logical or physical SAN resources are identified by an index. This method implies the existence of internal tables of the resources.

NOTE 2 These tables are only logically implied. No implementation constraint is intended

These tables include the collection of adapters 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, which raises the possible need to reassign indexes in implicit tables. But an application may expect to extract and maintain its own information about SAN resources, keyed to the API's implicit tables by the index. If the index assignment were to change without coordination, the information in the application's space would refer to different resources than the information in the API.

In the Common HBA API (see 4.4), the APIs implicit tables are static except at explicit synchronization calls. HBA_RefreshInformation updates all tables for a specified HBA. HBA_RefreshAdapterConfiguration was added in this standard to serve the same purpose for the table of adapters. This assures the application and the libraries remain consistent in the assignment of indexes to resources and also allows the application to control the amount of computation and communication invested in maintaining a contemporary view of the SAN configuration.

Although many kinds of dynamic changes may be detected promptly by the HBA software (e.g., by RSCN), the polled event reporting mechanism is unreliable. For an application to remain current, frequent polling of the entire name server is necessary, either by frequent calls to HBA_RefreshInformation or explicitly. Either is extremely costly in large SANs. To reduce the need for polling, the Common HBA API added two extensions that were consistent with the static table assumption: The HBA_STATUS_ERROR_STALE_DATA error to indicate that static tables are out of date (see 5.4.2), and semistatic tables (see 5.4.3).

The asynchronous event method in this standard provides a reliable notification of pending configuration changes; however, the earlier extensions have been retained for several reasons: They allow applications written for the Common HBA API to run on FC-HBA compliant libraries. They enable use of a simple application design that does not monitor asynchronous events. Finally, they may close possible implementation-dependent timing windows for applications that may concurrently process configuration change events and other events (including timers and operator input).

Finally, in this standard, new functions have been added that use WWNs rather than indexes into implicit tables to identify objects.

5.4.2 HBA_STATUS_ERROR_STALE_DATA

Rather than simply returning static obsolete information, a library shall return HBA_STATUS_ERROR_STALE_DATA. This shall be returned by any function that references an implicit table with a pending change for the calling application, and continues until HBA_RefreshInformation is called. Without changing the static table design, this prevents the application from unknowingly using the stale data. This error also notifies the application at the earliest point that the application would have accessed the stale data without requiring any extra overhead to monitor for changes.

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, if added to the tables prior to HBA_RefreshInformation, shall be assigned the least unassigned index. Calls that identify the number of some type of resource shall return the largest assigned index, even when some indexes are assigned to resources that no longer exist. The number of resources and the size of the table may therefore change without a call to a Refresh function, but these rules shall constrain such changes:

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

These rules imply that in systems that contain adapters from multiple vendors and allow dynamic HBA reconfiguration, it may not be possible for the wrapper library to assign contiguous HBA indexes to adapters 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 unstable or 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 where it would compromise this goal.

Implementations of the HBA API specified by this standard shall also be tolerant of certain concurrency issues. They may operate concurrently with other such applications, so the results of any given call may not be predicted based on any information gained from prior calls by the same application. Further, 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 int HBA_UINT32;  /* A 32 bit unsigned integer */

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

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

typedef HBA_UINT32 HBA_HANDLE;    /* opaque handle used to identify an HBA */

typedef struct HBA_wwn {HBA_UINT8 wwn[8];} HBA_WWN, *PHBA_WWN;/* An FC-FS Name_Identifier */
                                  /* The first byte of the Name_Identifier */
                                  /* shall be in the first byte of the array, */
                                  /* and successive bytes of the Name_Identifier */
                                  /* shall be in successive bytes of the array. */
```

6.2 Status Return Values

Functions that return an object of type HBA_STATUS shall set the value of that object to a value among those in 6.2.

```
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

/* WWN 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
```

```
/* Information has changed since the last call to HBA_RefreshInformation */
#define      HBA_STATUS_ERROR_STALE_DATA             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 and HBA_GetAdapterName. */
#define      HBA_STATUS_ERROR_INCOMPATIBLE           15

/* Multiple adapters have a matching WWN. */
/* This may occur if the NodeWWN of multiple adapters 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 Target FCID */
#define      HBA_STATUS_ERROR_TARGET_FCID            21

/* A persistent binding request included a bad FCP Target Node_Name */
#define      HBA_STATUS_ERROR_TARGET_NODE_WWN        22
```

```
/* A persistent binding request included a bad FCP Target 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 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 port */
#define       HBA_STATUS_ERROR_INCAPABLE              29

/* A SCSI function was requested at a time when issuing the requested command */
/* would cause a SCSI overlapped command condition (see SAM-3) */
#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
```

6.3 HBA Attributes

6.3.1 HBA Attribute Data Declarations

```
typedef struct HBA_AdapterAttributes {
      char                Manufacturer[64];
      char                SerialNumber[64];
      char                Model[256];
      char                ModelDescription[256];
      HBA_WWN             NodeWWN;
      char                NodeSymbolicName[256];
      char                HardwareVersion[256];
      char                DriverVersion[256];
      char                OptionROMVersion[256];
      char                FirmwareVersion[256];
      HBA_UINT32          VendorSpecificID;
      HBA_UINT32          NumberOfPorts;
      char                DriverName[256];
} HBA_ADAPTERATTRIBUTES, *PHBA_ADAPTERATTRIBUTES;
```

6.3.2 HBA Attribute Specifications

6.3.2.1 Compliance

Some HBA attributes shall be implemented as specified, and some shall either be implemented or given a value indicating they are not implemented (see annex A).

6.3.2.2 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
```

6.3.2.3 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
```

6.3.2.4 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 model.

NOTE 3 To allow complete representation in the CIM model of an HBA, the length of this attribute should not exceed 64 bytes.

Example:

```
HBA1040A
```

6.3.2.5 ModelDescription

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

Example:

```
Hot Biscuits Adapters Short Form
```

6.3.2.6 NodeWWN

NodeWWN shall be the Node_Name of this HBA. If an HBA has multiple end ports associated with more than one node, the value of NodeWWN shall be chosen from among the names of the associated nodes by vendor specific means.

6.3.2.7 NodeSymbolicName

NodeSymbolicName shall be an ASCII string not exceeding 256 bytes the value of which is the Fibre Channel Node Symbolic Name. In a Fabric, this shall be the same as the Node Symbolic Name registered with the name server.

6.3.2.8 HardwareVersion

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

NOTE 4 To allow complete representation in the CIM model of an HBA, the length of this attribute should not exceed 64 bytes.

6.3.2.9 DriverVersion

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

6.3.2.10 OptionROMVersion

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

6.3.2.11 FirmwareVersion

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

6.3.2.12 VendorSpecificID

VendorSpecificID shall have a vendor specific value.

6.3.2.13 NumberOfPorts

NumberOfPorts shall be the number of end ports on this HBA.

6.3.2.14 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
```

6.4 FC_Port Attributes

6.4.1 FC_Port Attribute Data Declarations

6.4.1.1 Port Type

Any data object of type HBA_PORTTYPE shall have a value defined in 6.4.1.1

```
typedef HBA_UINT32 HBA_PORTTYPE;

#define HBA_PORTTYPE_UNKNOWN            1      /* Unknown */
#define HBA_PORTTYPE_OTHER              2      /* Other */
#define HBA_PORTTYPE_NOTPRESENT         3      /* Not present */
#define HBA_PORTTYPE_NPORT              5      /* End port in a Fabric */
#define HBA_PORTTYPE_NLPORT             6      /* End port on a Public Loop */
#define HBA_PORTTYPE_FLPORT             7      /* Fabric port on a Loop */
#define HBA_PORTTYPE_FPORT              8      /* Fabric Port */
#define HBA_PORTTYPE_LPORT              20     /* End port on a Private Loop */
#define HBA_PORTTYPE_PTP                21     /* Point to Point */
```

6.4.1.2 Port State

Any data object of type HBA_PORTSTATE shall have a value defined in 6.4.1.2

```
typedefHBA_UINT32 HBA_PORTSTATE;

#define HBA_PORTSTATE_UNKNOWN           1      /* Unknown */
#define HBA_PORTSTATE_ONLINE            2      /* Operational */
#define HBA_PORTSTATE_OFFLINE           3      /* User Offline */
#define HBA_PORTSTATE_BYPASSED          4      /* Bypassed */
#define HBA_PORTSTATE_DIAGNOSTICS       5      /* In diagnostics mode */
#define HBA_PORTSTATE_LINKDOWN          6      /* Link Down */
#define HBA_PORTSTATE_ERROR             7      /* Port Error */
#define HBA_PORTSTATE_LOOPBACK          8      /* Loopback */
```

6.4.1.3 Port Speed

Any data object of type HBA_PORTSPEED shall have a value defined in 6.4.1.3

```
typedef HBA_UINT32 HBA_PORTSPEED;

#define HBA_PORTSPEED_UNKNOWN           0      /* Unknown - transceiver incapable
                                                      of reporting*/
#define HBA_PORTSPEED_1GBIT             1      /* 1 GBit/sec */
#define HBA_PORTSPEED_2GBIT             2      /* 2 GBit/sec */
#define HBA_PORTSPEED_10GBIT            4      /* 10 GBit/sec */
#define HBA_PORTSPEED_4GBIT             8      /* 4 GBit / sec */
#define HBA_PORTSPEED_NOT_NEGOTIATED    (1<<15)/* Speed not established*/
```

6.4.1.4 Class of Service

Any data object of type HBA_COS shall have a value as defined in FC-GS-4 for Class of Service - Format

```
typedef HBA_UINT32 HBA_COS;      /* See Class of Service - Format in FC-GS-4*/
```

6.4.1.5 FC-4 Types

Any data object of type HBA_FC4TYPES shall have a value as defined in FC-GS-4 for FC-4 TYPEs - Format

```
typedef struct HBA_fc4types {
      HBA_UINT8 bits[32];        /* See FC-4 TYPEs - Format in FC-GS-4 */
} HBA_FC4TYPES, *PHBA_FC4TYPES;
```

6.4.1.6 FC_Port Attributes

Any data object of type HBA_PORTATTRIBUTES shall have the format defined in 6.4.1.6

```
typedef struct HBA_PortAttributes {
      HBA_WWN             NodeWWN;
      HBA_WWN             PortWWN;
      HBA_UINT32          PortFcId;
      HBA_PORTTYPE        PortType;
      HBA_PORTSTATE       PortState;
      HBA_COS             PortSupportedClassofService;
      HBA_FC4TYPES        PortSupportedFc4Types;
      HBA_FC4TYPES        PortActiveFc4Types;
      char                PortSymbolicName[256];
      char                OSDeviceName[256];
      HBA_PORTSPEED       PortSupportedSpeed;
      HBA_PORTSPEED       PortSpeed;
      HBA_UINT32          PortMaxFrameSize;
      HBA_WWN             FabricName;
      HBA_UINT32          NumberofDiscoveredPorts;
} HBA_PORTATTRIBUTES, *PHBA_PORTATTRIBUTES;
```

6.4.2 FC_Port Attribute Specifications

6.4.2.1 Compliance

Some FC_Port Attributes shall be implemented as specified, and some shall either be implemented or given a value indicating they are not implemented (see annex A).

6.4.2.2 NodeWWN

NodeWWN shall be the Fibre Channel Node_Name associated with this FC_Port.

6.4.2.3 PortWWN

PortWWN shall be the Fibre Channel Port Name of this FC_Port.

6.4.2.4 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-4). In a Fabric, this shall be the same as the entry registered with the name server.

6.4.2.5 PortFcId

PortFcId shall be an unsigned integer the value of which is the current Fibre Channel address identifier of the FC_Port. The first byte of the address identifier shall be stored in the high order byte of PortFcId, and successive bytes of the address identifier shall be stored in successively lower order bytes of PortFcId. The lowest order byte of PortFcId shall be zero.

6.4.2.6 PortType

PortType shall be an enumerated type that identifies the General Services Port Type, sometimes elaborated by link topology, that the FC_Port is currently operating in FC-GS-4. It shall have a value defined in 6.4.1.1

6.4.2.7 PortState

PortState shall be an integer the value of which indicates the current state of the FC_Port. It shall have a value defined in 6.4.1.2. The sequence and timing of FC_Port states exhibited as an FC_Port experiences errors or transient conditions is vendor specific.

6.4.2.8 PortSupportedClassofService

PortSupportedClassofService shall identify the supported classes of service of this FC_Port. It shall have a value as defined in FC-GS-4 for Class of Service - Format.

6.4.2.9 PortSupportedFc4Types

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

The first word of the FC-4 TYPEs structure defined by GS-4 shall be stored in bytes zero through three of the HBA_FC4TYPES bits array, and successive words of the FC-4 TYPEs structure defined by GS-4 shall be stored in successively higher numbered four byte groups of the HBA_FC4TYPES bits array. The lowest order byte of each word shall be stored in the lowest numbered byte of the group of four bytes in which the word is stored, and successively higher order bytes of each word shall be stored in successively higher numbered bytes of the group of bytes in which the word is stored.

6.4.2.10 PortActiveFc4Types

PortActiveFc4Types shall identify the FC-4 types that have been determined to be currently available from this end port. It shall be zero if the port is not an end port. It shall have a value as defined in FC-GS-4 for FC-4 TYPEs - Format.

The first word of the FC-4 TYPEs structure defined by GS-4 shall be stored in bytes zero through three of the HBA_FC4TYPES bits array, and successive words of the FC-4 TYPEs structure defined by GS-4 shall be stored in successively higher numbered four byte groups of the HBA_FC4TYPES bits array. The lowest order byte of each word shall be stored in the lowest numbered byte of the group of four bytes in which the word is stored, and successively higher order bytes of each word shall be stored in successively higher numbered bytes of the group of bytes in which the word is stored.

6.4.2.11 PortSupportedSpeed

PortSupportedSpeed shall identify the signalling bit rates at which this FC_Port may operate. It shall have a value defined in 6.4.1.3. It may identify multiple speeds.

6.4.2.12 PortSpeed

PortSpeed shall identify the signalling bit rate at which this FC_Port is currently operating. It shall have a value defined in 6.4.1.3. It shall indicate only a single speed.

6.4.2.13 PortMaxFrameSize

PortMaxFrameSize shall have value equal to the maximum frame size in bytes supported by this FC_Port.

6.4.2.14 OSDeviceName

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

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

a) A non-null end port OSDeviceName shall be provided if, and only 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 satisfy 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 satisfy 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).
d) If no name satisfies rule a), the OSDeviceName shall be a zero length ASCII string.

Table 1 — Preferred format for FC_Port OSDeviceName

OS	**Preferred format** [a]	
	Local adapter port	**Discovered port**
AIX	**/dev/fscsi***n*	(zero length string)
Linux	**/dev/***name*	(zero length string)
Solaris	**/devices/***localportname*	**c***X***:***pwwn* (the attachment point ID)
Windows	**\\.\Scsi***n***:**	(zero length string)

[a] In end port name format samples, text appearing in bold weight shall appear in the indicated position exactly 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 n represents any decimal number and may be more than one digit. Normal text in parentheses is descriptive, not format sample.

6.4.2.15 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 FC_Ports this value shall be zero.

6.4.2.16 FabricName

FabricName shall have value equal to the Name_Identifier for the Fabric to which the FC_Port is attached, if known.

6.5 End Port Statistics

6.5.1 End Port Statistics Data Declarations

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

typedef struct HBA_PortStatistics {
      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_PORTSTATISTICS, *PHBA_PORTSTATISTICS;

/* Statistical counters for FC-4 protocols */

typedef struct HBA_FC4Statistics {
      HBA_INT64           InputRequests;
      HBA_INT64           OutputRequests;
      HBA_INT64           ControlRequests;
      HBA_INT64           InputMegabytes;
      HBA_INT64           OutputMegabytes;
} HBA_FC4STATISTICS, *PHBA_FC4STATISTICS;
```

6.5.2 End Port Statistics Attribute Specifications

6.5.2.1 Compliance

Some end port statistics shall be implemented as specified, and some shall either be implemented or given value FFFF FFFF FFFF FFFFh indicating they are not implemented (see annex A).

Statistics counters shall be 64-bit unsigned integers that shall wrap to zero on exceeding 7FFF FFFF FFFF FFFFh. They shall not be reset during normal operation so traffic rates may be determined by the difference of time and counter values at two successive calls, with appropriate measures to deal with counter wrap.

6.5.2.2 SecondsSinceLastReset

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

6.5.2.3 TxFrames

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

6.5.2.4 RxFrames

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

6.5.2.5 TxWords

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

6.5.2.6 RxWords

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

6.5.2.7 LIPCount

LIPCount shall have value equal to the number of LIP events that have occurred on a arbitrated loop.

6.5.2.8 NOSCount

NOSCount shall have value equal to the number of NOS events that have occurred on the switched Fabric.

6.5.2.9 ErrorFrames

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

6.5.2.10 DumpedFrames

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

6.5.2.11 LinkFailureCount

LinkFailureCount shall have 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).

6.5.2.12 LossOfSyncCount

LossOfSyncCount shall have 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).

6.5.2.13 LossOfSignalCount

LossOfSignalCount shall have 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).

6.5.2.14 PrimitiveSeqProtocolErrCount

Primitive Sequence Protocol Error Count shall have 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).

6.5.2.15 InvalidTxWordCount

InvalidTxWordCount shall have 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).

6.5.2.16 Invalid CRC Count

InvalidCRCCount shall have 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).

6.5.2.17 InputRequests

InputRequests shall have value equal to the number of FC-4 operations causing FC-4 data input. Some single FC-4 requests may cause both input and output of data (e.g., Bidirectional SCSI commands in FCP). If these requests occur, they shall be counted in both InputRequests and OutputRequests. This admits the possibility that the sum of InputRequests and OutputRequests may exceed the total number of requests.

6.5.2.18 OutputRequests

OutputRequests shall have value equal to the number of FC-4 operations causing FC-4 data output. Some single FC-4 requests may cause both input and output of data (e.g., Bidirectional SCSI commands in FCP). If these requests occur, they shall be counted in both InputRequests and OutputRequests. This admits the possibility that the sum of InputRequests and OutputRequests may exceed the total number of requests.

6.5.2.19 ControlRequests

ControlRequests shall have value equal to the number of FC-4 operations that are not intended to cause FC-4 data movement.

6.5.2.20 InputMegabytes

InputMegabytes shall have value equal to the number of megabytes (mega = 1 000 000) of FC-4 data input.

6.5.2.21 OutputMegabytes

OutputMegabytes shall have value equal to the number of megabytes (mega = 1 000 000) of FC-4 data output.

6.6 FCP_Port Attributes (see FCP-2)

6.6.1 FCP_Port Attribute Data Declarations

6.6.1.1 HBA_FCPBINDINGTYPE

```
/* A bit mask of Phase 1 persistent binding capabilities */

typedef enum HBA_fcpbindingtype {TO_D_ID, TO_WWN,TO_OTHER} HBA_FCPBINDINGTYPE;
```

6.6.1.2 HBA_BIND_CAPABILITY

Any data object of type HBA_BIND_CAPABILITY shall have a value defined in 6.6.1.2

```
typedef HBA_UINT32 HBA_BIND_CAPABILITY;

#define HBA_CAN_BIND_TO_D_ID 0x0001
#define HBA_CAN_BIND_TO_WWPN 0x0002
#define HBA_CAN_BIND_TO_WWNN 0x0004
#define HBA_CAN_BIND_TO_LUID 0x0008
#define HBA_CAN_BIND_ANY_LUNS 0x400
#define HBA_CAN_BIND_TARGETS 0x800
#define HBA_CAN_BIND_AUTOMAP 0x1000
#define HBA_CAN_BIND_CONFIGURED 0x2000
```

6.6.1.3 HBA_BIND_TYPE

Any data object of type HBA_BIND_TYPE shall have a value defined in 6.6.1.3

```
typedef HBA_UINT32 HBA_BIND_TYPE;

#define HBA_BIND_TO_D_ID 0x0001
#define HBA_BIND_TO_WWPN 0x0002
#define HBA_BIND_TO_WWNN 0x0004
#define HBA_BIND_TO_LUID 0x0008
#define HBA_BIND_TARGETS 0x800
```

6.6.1.4 HBA_LUID

```
typedef struct HBA_LUID {
      char                buffer[256];
} HBA_LUID, *PHBA_LUID;
```

6.6.1.5 HBA_ScsiId

```
typedef struct HBA_ScsiId {
      char                OSDeviceName[256];
      HBA_UINT32          ScsiBusNumber;
      HBA_UINT32          ScsiTargetNumber;
      HBA_UINT32          ScsiOSLun;
} HBA_SCSIID, *PHBA_SCSIID;
```

6.6.1.6 HBA_FcpId

```
typedef struct HBA_FcpId {
      HBA_UINT32          FcId;
      HBA_WWN             NodeWWN;
      HBA_WWN             PortWWN;
      HBA_UINT64          FcpLun;
} HBA_FCPID, *PHBA_FCPID;
```

6.6.1.7 Composite types

```
typedef struct HBA_FcpScsiEntry {
      HBA_SCSIID          ScsiId;
      HBA_FCPID           FcpId;
} HBA_FCPSCSIENTRY, *PHBA_FCPSCSIENTRY;
```

```
typedef struct HBA_FcpScsiEntryV2 {
      HBA_SCSIID ScsiId;
      HBA_FCPID FcpId;
      HBA_LUID LUID;
} HBA_FCPSCSIENTRYV2, *PHBA_FCPSCSIENTRYV2;

typedef struct HBA_FCPTargetMapping {
      HBA_UINT32          NumberOfEntries;
      HBA_FCPSCSIENTRY    entry[1];           /* Variable length array containing
                                              mappings*/
} HBA_FCPTARGETMAPPING, *PHBA_FCPTARGETMAPPING;

typedef struct HBA_FCPTargetMappingV2 {
      HBA_UINT32 NumberOfEntries;
      HBA_FCPSCSIENTRYV2 entry[1];      /* Variable length array containing
                                        mappings*/
} HBA_FCPTARGETMAPPINGV2, *PHBA_FCPTARGETMAPPINGV2;

typedef struct HBA_FCPBindingEntry {
      HBA_FCPBINDINGTYPE  type;
      HBA_SCSIID          ScsiId;
      HBA_FCPID           FcpId;
      HBA_UINT32          FcId;
} HBA_FCPBINDINGENTRY, *PHBA_FCPBINDINGENTRY;

typedef struct HBA_FCPBinding {
      HBA_UINT32                NumberOfEntries;
      HBA_FCPBINDINGENTRY       entry[1];     /* Variable length array */
} HBA_FCPBINDING, *PHBA_FCPBINDING;

typedef struct HBA_FCPBindingEntry2 {
      HBA_BIND_TYPE       type;
      HBA_SCSIID          ScsiId;
      HBA_FCPID           FcpId;
      HBA_LUID            LUID;
      HBA_STATUS          Status;
} HBA_FCPBINDINGENTRY2, *PHBA_FCPBINDINGENTRY2;

typedef struct HBA_FcpBinding2 {
      HBA_UINT32                NumberOfEntries;
      HBA_FCPBINDINGENTRY2      entry[1]; /* Variable length array */
} HBA_FCPBINDING2, *PHBA_FCPBINDING2;
```

6.6.2 Target Mapping and Persistent Binding Attribute Specifications

6.6.2.1 HBA_FCPBINDINGTYPE

NOTE 5 HBA_FCPBINDINGTYPE has been retained in this standard for compatibility with HBA API Phase 1 (see FC-MI).

The value of a data object of type HBA_FCPBINDINGTYPE shall be as indicated in its declaration. Symbolic constant TO_OTHER shall be used in Phase 1 compatible HBA_GetPersistentBinding functions to indicate binding types not defined in Phase 1. Phase 2 functions, in order that they may distinguish several additional types of persistent bindings, do not specify this type.

6.6.2.2 HBA_BIND_CAPABILITY

A data object of type HBA_BIND_CAPABILITY shall represent the ability of an HBA to provide 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 HBA_BIND_CAPABILITY shall have a value equal to the bit-wise OR of zero or more symbolic constants declared in 6.6.1.2 and defined in 6.6.3

6.6.2.3 HBA_BIND_TYPE

A data object of type HBA_BIND_TYPE shall indicate a set of Persistent Binding features that are relevant to a specific Persistent Binding. Any data object of type HBA_BIND_TYPE shall have a value equal to the bit-wise OR of zero or more symbolic constants declared in 6.6.1.3 and defined in 6.6.4

6.6.2.4 HBA_LUID

A data object of type HBA_LUID shall have value equal to an Identification Descriptor from the Vital Products Data Device Identification Page (VPD Page 83h) returned by a logical unit in reply to a SCSI INQUIRY command as specified in SPC-3 with further constraints specified in this subclause. Its length shall be 256 bytes or less. Its Association value shall be device association (zero) and its Identifier Type shall be one of Vendor Specific (zero), T10 vendor identification (one), EUI-64 (two) or Name_Identifier as defined in FC-FS (three). An Identification Descriptor of Identifier Type two or three 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. One should be used only if it is the only alternative, or its persistence and uniqueness are known by the local administration to be sufficient.

6.6.2.5 HBA_SCSIID

A data object of type HBA_SCSIID shall encapsulate an operating system identification of a SCSI logical unit. The value of its OSDeviceName field shall be as specified in 6.6.2.11. The value of its ScsiBusNumber field shall be as specified in 6.6.2.12. The value of its ScsiTargetNumber field shall be as specified in 6.6.2.13. The value of its ScsiOSLun field shall be as specified in 6.6.2.14.

NOTE 6 Most versions of Windows and Unix and their application programs identify storage resources via an abstraction of the classic SCSI Parallel Interface architecture (see SAM-3): A resource is identified 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, a character string.

6.6.2.6 HBA_FCPID

A data object of type HBA_FCPID shall represent the identification of a SCSI logical unit on an FCP-2 service delivery system. The value of its FcId field shall be as specified in 6.6.2.9. The value of its NodeWWN field shall be as specified in 6.6.2.7. The value of its PortWWN field shall be as specified in 6.6.2.8. The value of its FcpLun field shall be as specified in 6.6.2.10.

6.6.2.7 NodeWWN

Within the context of any FCP attribute data structures defined in 6.6.1, the value of a field with symbolic name NodeWWN shall be zero or the Node_Name of an FCP-2 Target device (see SAM-3).

Within the context of a target mapping returned from an HBA API function, NodeWWN shall be the Node_Name of the FCP-2 Target device that is represented in the mapping.

6.6.2.8 PortWWN

Within the context of any FCP attribute data structures defined in 6.6.1, the value of a field with symbolic name PortWWN shall be zero or the N_Port_Name of an FCP-2 Target device (see SAM-3).

Within the context of a target mapping returned from an HBA API function, PortWWN shall be the Port Name of the FCP-2 Target device that is represented in the mapping.

6.6.2.9 FcId

Within the context of FCP attribute data structures defined in 6.6.1, the value of a field with symbolic name FcId shall be zero or the N_Port_ID of an FCP-2 Target device (see SAM-3).

6.6.2.10 FcpLun

Within the context of any FCP attribute data structures defined in 6.6.1, the value of a field with symbolic name FcpLun shall be zero or the 64-bit SCSI LUN of a SCSI logical unit within an FCP-2 Target device (see SAM-3).

Within the context of a target mapping returned from an HBA API function, if the mapping is to a specific logical unit, the value of FcpLun shall be the 64-bit SCSI LUN of the logical unit that is mapped, or if the mapping is to a target device, the value of FcpLun shall be the logical unit not specified extended address method LUN.

Byte zero of a SCSI LUN shall be stored as the highest order eight bits of the value of an FcpLun field, and successive bytes of the SCSI LUN shall be stored as successively lower order groups of eight bits of the FcpLun field.

6.6.2.11 OSDeviceName

Within the context of FCP attribute data structures defined in 6.6.1, the value of a field with symbolic name OSDeviceName shall be an ASCII string that is null or the name by which the operating system represents a SCSI Logical Unit (see SAM-3) to application programs, if known. This attribute is an ASCII string with length from 1 to 256 bytes.

If an OSDeviceName is provided by the HBA API in an HBA_ScsiId structure within an HBA_FCPTargetMapping or HBA_FCPTargetMappingv2 structure, it shall comply with these rules:

a) A non-null logical unit OSDeviceName shall be provided if, and only 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 2 and also satisfy 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 2 but there are names that satisfy 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).
d) If no name satisfies rule a), the OSDeviceName shall be a zero length ASCII string.

Table 2 — Preferred format for logical unit OSDeviceName

OS	Preferred format for logical unit type [a]			
	disk/optical	**cd-rom**	**tape**	**changer**
AIX	**/dev/hdisk***n* (disk) or **/dev/omd***n* (optical)	**/dev/cd***n*	**/dev/rmt***n*	(zero length string)
Linux	**/dev/sd***n*	**/dev/sr***n*	**/dev/st/***n*	(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/***n***n**	(zero length string)
Windows	**\\.\PHYSICALDRIVE***n*	**\\.\CDROM***n*	**\\.\TAPE***n*	**\\.\CHANGER***n*

[a] In logical unit name format samples, text appearing in bold weight shall appear in the indicated position exactly 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 n represents any decimal number and may be more than one digit. 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 Solaris convention, rdsk shall be used to indicate the raw device and s2 shall be used to reference the unpartitioned disk. Should other conventional formats be established for representing these characteristics, the new conventions shall also be considered preferred formats.

6.6.2.12 ScsiBusNumber

Within the context of any FCP attribute data structures defined in 6.6.1, the value of a field with symbolic name ScsiBusNumber shall be 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-3 and relevant OS documentation).

Within the context of a target mapping returned from an HBA 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 value of the ScsiBusNumber shall be its registered local bus number.

6.6.2.13 ScsiTargetNumber

Within the context of any FCP attribute data structures defined in 6.6.1, the value of a field with symbolic name ScsiTargetNumber shall be 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-3 and relevant OS documentation).

Within the context of a target mapping returned from an HBA API function, the value of ScsiTargetNumber shall be the OS Target ID of the device that is mapped.

6.6.2.14 ScsiOSLun

Within the context of any FCP attribute data structures defined in 6.6.1, the value of a field with symbolic name ScsiOSLun shall be 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-3 and relevant OS documentation).

Within the context of a target mapping returned from an HBA API function, if the mapping is to a specific logical unit, the value of ScsiOSLun shall be the OS LUN of the logical unit that is mapped, or if the mapping is to a target device, the value of ScsiOSLun shall be vendor specific.

6.6.3 Persistent Binding Capabilities

6.6.3.1 Persistent Binding Capability: HBA_CAN_BIND_TO_D_ID

The Persistent Binding capability HBA_CAN_BIND_TO_D_ID shall indicate the ability of an HBA to accept a Persistent Binding that identifies the Fibre Channel target port by its address identifier.

6.6.3.2 Persistent Binding Capability: HBA_CAN_BIND_TO_WWPN

The Persistent Binding capability HBA_CAN_BIND_TO_WWPN shall indicate the ability of an HBA to accept a Persistent Binding that identifies the Fibre Channel target port by its WWPN.

6.6.3.3 Persistent Binding Capability: HBA_CAN_BIND_TO_WWNN

The Persistent Binding capability HBA_CAN_BIND_TO_WWNN shall indicate the ability of an HBA to accept a Persistent Binding that identifies a Fibre Channel target device (not a target port) by its World Wide Node Name (WWNN). Its ambiguity with respect to multi-port devices is intentional, being left for the HBA and / or fabric to resolve in vendor specific manner.

6.6.3.4 Persistent Binding Capability: HBA_CAN_BIND_TO_LUID

The Persistent Binding capability HBA_CAN_BIND_TO_LUID shall indicate the ability of an HBA to accept a Persistent Binding that identifies the Fibre Channel target logical unit by the value of one of its device-associated Identification Descriptors (LUID).

6.6.3.5 Persistent Binding Capability: HBA_CAN_BIND_ANY_LUNS

The Persistent Binding capability HBA_CAN_BIND_ANY_LUNS shall indicate the ability of an HBA to accept Persistent Binding settings that independently specify both the ScsiOSLuns and FcpLuns.

An HBA that does not express the HBA_CAN_BIND_ANY_LUNS capability may require that all Persistent Binding settings preserve the groupings of logical units into devices, i.e., 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 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 target port identified by the persistent bindings is different.

NOTE 7 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.

6.6.3.6 Persistent Binding Capability: HBA_CAN_BIND_TARGETS

The Persistent Binding capability HBA_CAN_BIND_TARGETS shall indicate the ability of an HBA to interpret a single Persistent Binding setting as direction to automatically generate Target Mappings for all LUNs offered by the target device identified in the HBA_FCPID of the persistent binding setting to LUNS subordinate to the bus and target numbers identified in the HBA_SCSIID of the persistent binding.

6.6.3.7 Persistent Binding Capability: HBA_CAN_ BIND_AUTOMAP

The Persistent Binding capability HBA_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 by means specified in this standard or otherwise.

NOTE 8 This capability is sometimes described as HBA-based LUN Masking.

6.6.3.8 Persistent Binding Capability: HBA_CAN_BIND_CONFIGURED

The Persistent Binding capability HBA_CAN_BIND_CONFIGURED shall indicate the ability of an HBA to accept the Persistent Binding configuration functions HBA_SetPersistentBindingV2, HBA_RemovePersistentBinding, and HBA_RemoveAllPersistentBindings.

An HBA that does not express this capability may provide only vendor specific or automatically generated configuration of persistent bindings.

6.6.4 Persistent Binding Setting Types

6.6.4.1 Persistent Binding Type: HBA_BIND_TO_D_ID

If a Persistent Binding setting includes this feature in its type, the setting shall identify the Fibre Channel target port by its FcId field. The PortWWN, NodeWWN, and LUID fields shall be ignored.

If a Persistent Binding setting includes more than one of HBA_BIND_TO_D_ID, HBA_BIND_TO_WWPN, HBA_BIND_TO_WWNN, AND HBA_BIND_TO_LUID in its type, that setting shall be rejected.

6.6.4.2 Persistent Binding Type: HBA_BIND_TO_WWPN

If a Persistent Binding setting includes this feature in its type, the setting shall identify the Fibre Channel target port by its PortWWN field. The FcId, NodeWWN, and LUID fields shall be ignored.

If a Persistent Binding setting includes more than one of HBA_BIND_TO_D_ID, HBA_BIND_TO_WWPN, HBA_BIND_TO_WWNN, AND HBA_BIND_TO_LUID in its type, that setting shall be rejected.

6.6.4.3 Persistent Binding Type: HBA_BIND_TO_WWNN

If a Persistent Binding setting includes this feature in its type, the setting shall identify the Fibre Channel target device by its NodeWWN field. The FcId, PortWWN, and LUID fields shall be ignored. The HBA shall choose by vendor specific means an appropriate target port associated with the identified target device.

If a Persistent Binding setting includes more than one of HBA_BIND_TO_D_ID, HBA_BIND_TO_WWPN, HBA_BIND_TO_WWNN, AND HBA_BIND_TO_LUID in its type, that setting shall be rejected.

6.6.4.4 Persistent Binding Type: HBA_BIND_TO_LUID

If a Persistent Binding setting includes this feature in its type, the setting shall identify the Fibre Channel target logical unit by its LUID field. The FcpLun, FcId, PortWWN, and NodeWWN fields shall be ignored.

If a Persistent Binding setting includes more than one of HBA_BIND_TO_D_ID, HBA_BIND_TO_WWPN, HBA_BIND_TO_WWNN, AND HBA_BIND_TO_LUID in its type, that setting shall be rejected.

6.6.4.5 Persistent Binding Type: HBA_BIND_TARGETS

If a Persistent Binding setting includes this feature in its type, Target Mappings shall be automatically generated from OS LUNs on the controller and target indicated by the HBA_SCSIID of the persistent binding to all logical units on the FCP target port indicated by HBA_FCPID of the persistent binding, up to the capacity of the OS target implementation. The LUNs in the HBA_FCPID and the HBA_SCSIID shall be ignored.

If a setting does not include this feature in its type, the setting shall be treated as a persistent binding of the specified OS logical unit to the specified FCP logical unit. The HBA may not have the HBA_CAN_BIND_ANY_LUNS capability, in which case the Persistent Binding settings that may be configured may be restricted to those that preserve logical unit groupings within targets as described in 6.6.3.5.

6.7 SB Attributes

```
typedef struct HBA_SBDevId {
      char                OSDeviceName[256];
} HBA_SBDEVID, *PHBA_SBDEVID;

typedef struct HBA_SBId {
      HBA_UINT32          FcId;
      HBA_WWN             NodeWWN;
      HBA_WWN             PortWWN;
      HBA_UINT32          SBDeviceIdentifier;
} HBA_SBID, *PHBA_SBID;

typedef struct HBA_Ned {
      HBA_UINT32          NEDWord0;
      HBA_UINT32          NelId[7];                 /* Node Element Identifier*/
} HBA_NED, *PHBA_NED;                         /* Node Element Descriptor*/

typedef struct HBA_DeviceSelfDesc {
      HBA_NED             TokenNED;
      HBA_NED             DeviceNED;
} HBA_DEVICESELFDESC, *PHBA_DEVICESELFDESC;

typedef struct HBA_SBDevEntry {
      HBA_SBDEVID         SBDevId;
      HBA_SBID            SBId;
      HBA_DEVICESELFDESC  DeviceSelfDesc;           /* Device Self Description Data*/
} HBA_SBDEVENTRY, *PHBA_SBDEVENTRY;

typedef struct HBA_SBTargetMapping {
      HBA_UINT32          NumberOfEntries;
      HBA_SBDEVENTRY      entry[1];           /* Variable length array containing mappings*/
} HBA_SBTARGETMAPPING, *PHBA_SBTARGETMAPPING;
```

```
typedef struct HBA_SBStatistics {
      HBA_INT32           SSCHRSCHCount;
      HBA_INT32           SampleCount;
      HBA_INT32           DeviceConnectTime;
      HBA_INT32           FunctionPendingTime;
      HBA_INT32           DeviceDisconnectTime;
      HBA_INT32           ControlUnitQueuingTime;
      HBA_INT32           DeviceActiveOnlyTime;
      HBA_INT32           Reserved;
      HBA_INT32           Reserved;
} HBA_SBSTATISTICS, *PHBA_SBSTATISTICS;

typedef struct HBA_SBDskCapacity {
      HBA_INT32           SCSIFormatLBA;             /* SCSI Read Capacity Format */
      HBA_INT32           SCSIFormatBlkLen;          /* SCSI Read Capacity Format */
      HBA_INT32           SBDskNumberOfCylinders;    /* cyls*/
      HBA_INT32           SBDskTracksPerCylinder;    /* tracks per cyl  */
      HBA_INT32           SBDskMaxUsableTrackLen;    /* usable track capacity */
} HBA_SBDSKCAPACITY, *PHBA_SBDSKCAPACITY;
```

6.8 FC-3 Management Attributes

6.8.1 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;
```

6.8.2 FC-3 Management Attribute Overview

NOTE 9 Although the HBA_MgmtInfo structure closely resembles the Specific Identification Data in an RNID Accept with Node Identification Data Format DFh (see FC-FS), it is different. First, it includes only 8 bytes of the initial 16 bytes of the Specific Identification Data. Further, the names of the fields in this structure reflect an earlier version of the reply to the RNID ELS. RNID was significantly redefined in FC-FS after the predecessor to this standard had stabilized.

6.8.3 FC-3 Management Attribute Specifications

6.8.3.1 Compliance

Some FC-3 management attributes shall be implemented as specified, and some shall either be implemented or given a value indicating they are not implemented (see annex A).

6.8.3.2 WWN

The value of the WWN field of a data structure of type HBA_MGMTINFO shall be the value of the first eight bytes of the initial 16 bytes of the Specific Identification Data in an RNID Accept with Node Identification Data Format DFh (see FC-FS). This is vendor specific data.

6.8.3.3 unittype

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

6.8.3.4 PortId

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

6.8.3.5 NumberOfAttachedNodes

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

6.8.3.6 IPVersion

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

6.8.3.7 UDPPort

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

6.8.3.8 IPAddress

The value of the IPAddress field of a data structure of type HBA_MGMTINFO shall be the value of the IP address field of the Specific Identification Data in an RNID Accept with Node Identification Data Format DFh (see FC-FS), 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.

6.8.3.9 TopologyDiscoveryFlags

The value of the TopologyDiscoveryFlags field of a data structure of type HBA_MGMTINFO shall be the value of the vendor specific field in word 12 of the Specific Identification Data in an RNID Accept with Node Identification Data Format DFh (see FC-FS).

6.9 Polled Event Notification Attributes

6.9.1 Polled Event Data Declarations

6.9.1.1 Polled Event Codes

```
#define      HBA_EVENT_LIP_OCCURRED           1
#define      HBA_EVENT_LINK_UP                2
#define      HBA_EVENT_LINK_DOWN              3
#define      HBA_EVENT_LIP_RESET_OCCURRED     4
#define      HBA_EVENT_RSCN                   5
#define      HBA_EVENT_PROPRIETARY            0xFFFF
```

6.9.1.2 Polled Event Data Structure Declarations

```
typedef struct HBA_Link_EventInfo {
      HBA_UINT32   PortFcId;          /* end port at which this event occurred */
      HBA_UINT32   Reserved[3];
} HBA_LINK_EVENTINFO, *PHBA_LINK_EVENTINFO;

typedef struct HBA_RSCN_EventInfo {
      HBA_UINT32   PortFcId;          /* end port at which this event occurred */
      HBA_UINT32   NPortPage;         /* Reference FC-FS for RSCN ELS
                                         Affected Port_ID Pages*/
      HBA_UINT32   Reserved[2];
} HBA_RSCN_EVENTINFO, *PHBA_RSCN_EVENTINFO;

typedef struct HBA_Pty_EventInfo {
      HBA_UINT32   PtyData[4];        /* Proprietary data */
} HBA_PTY_EVENTINFO, *PHBA_PTY_EVENTINFO;

typedef struct HBA_EventInfo {
      HBA_UINT32   EventCode;
      union {
            HBA_LINK_EVENTINFO        Link_EventInfo;
            HBA_RSCN_EVENTINFO        RSCN_EventInfo;
            HBA_PTY_EVENTINFO         Pty_EventInfo;
      } Event;
} HBA_EVENTINFO, *PHBA_EVENTINFO;
```

6.9.2 Polled Event Attribute Specifications

6.9.2.1 EventCode

EventCode shall contain an event code describing an event reported by the polled event API (see 6.9.1.1).

6.10 Asynchronous Event Notification Attributes

6.10.1 Asynchronous Event Data Declarations

6.10.1.1 Callback Handle

```
typedef void *HBA_CALLBACKHANDLE;
```

6.10.1.2 HBA Add Category Event Type

```
#define       HBA_EVENT_ADAPTER_ADD             0x101
```

6.10.1.3 HBA Category Event Types

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

6.10.1.4 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
```

6.10.1.5 Port Statistics Category Event Types

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

6.10.1.6 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
```

6.10.1.7 Link Category Event Types

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

6.10.2 Asynchronous Event Attribute Specifications

6.10.2.1 EventType

EventType shall contain an event type describing an event reported by the asynchronous event API (see 6.10.1).

a) The value of EventType shall be HBA_EVENT_ADAPTER_ADD to indicate that an HBA supported by the HBA API has been added to the local system.
b) The value of EventType shall be HBA_EVENT_ADAPTER_REMOVE to indicate that an HBA supported by the HBA API has been removed from the local system.
c) The value of EventType shall be HBA_EVENT_ADAPTER_CHANGE to indicate that there has been a configuration change to an HBA on the local system supported by the HBA API.
d) The value of EventType shall be HBA_EVENT_PORT_OFFLINE to indicate that an HBA on the local system supported by the HBA API has stopped providing communication.
e) The value of EventType shall be HBA_EVENT_PORT_ONLINE to indicate that an HBA on the local system supported by the HBA API has restarted providing communication.
f) The value of EventType shall be HBA_EVENT_PORT_NEW_TARGETS to indicate that an HBA on the local system supported by the HBA API has added FCP target devices to its discovered ports.
g) The value of EventType shall be HBA_EVENT_PORT_FABRIC to indicate that an HBA on the local system supported by the HBA API has received an RSCN ELS.

h) The value of EventType shall be HBA_EVENT_PORT_STAT_THRESHOLD to indicate that a statistical counter has reached a registered level.
i) The value of EventType shall be HBA_EVENT_PORT_STAT_GROWTH to indicate that a statistical counter has increased at a rate equal to or in excess of a registered rate.
j) The value of EventType shall be HBA_EVENT_TARGET_OFFLINE to indicate that operational use of an FCP target port supported by the HBA API has become impossible.
k) The value of EventType shall be HBA_EVENT_TARGET_ONLINE to indicate that operational use of an FCP target port supported by the HBA API has been restored.
l) The value of EventType shall be HBA_EVENT_TARGET_REMOVED to indicate that an FCP target port supported by the HBA API has been removed from the fabric.
m) The value of EventType shall be HBA_EVENT_LINK_INCIDENT to indicate that an HBA on the local system supported by the HBA API has received an RLIR ELS.

6.11 Library Attributes

6.11.1 Library Attribute Data Declarations

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

```
The following are prototypes for the functions specified in both HBA API Phase 1 and 2 librar-
ies:

typedef HBA_UINT32  (* HBAGetVersionFunc)();
typedef HBA_STATUS  (* HBALoadLibraryFunc)();
typedef HBA_STATUS  (* HBAFreeLibraryFunc)();
typedef HBA_UINT32  (* HBAGetNumberOfAdaptersFunc)();
typedef HBA_STATUS  (* HBAGetAdapterNameFunc)(HBA_UINT32, char *);
typedef HBA_HANDLE  (* HBAOpenAdapterFunc)(char *);
typedef void        (* HBACloseAdapterFunc)(HBA_HANDLE);
typedef HBA_STATUS  (* HBAGetAdapterAttributesFunc)
                    (HBA_HANDLE, HBA_ADAPTERATTRIBUTES *);
typedef HBA_STATUS  (* HBAGetAdapterPortAttributesFunc)
                    (HBA_HANDLE, HBA_UINT32, HBA_PORTATTRIBUTES *);
typedef HBA_STATUS  (* HBAGetPortStatisticsFunc)
                    (HBA_HANDLE, HBA_UINT32, HBA_PORTSTATISTICS *);
typedef HBA_STATUS  (* HBAGetDiscoveredPortAttributesFunc)
                    (HBA_HANDLE, HBA_UINT32, HBA_UINT32,HBA_PORTATTRIBUTES *);
typedef HBA_STATUS  (* HBAGetPortAttributesByWWNFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_PORTATTRIBUTES *);
typedef HBA_STATUS(* HBASendCTPassThruFunc)
                    (HBA_HANDLE, void *, HBA_UINT32, void *, HBA_UINT32);
typedef void        (* HBARefreshInformationFunc)(HBA_HANDLE);
typedef void        (* HBAResetStatisticsFunc)(HBA_HANDLE, HBA_UINT32);
typedef HBA_STATUS  (* HBAGetFcpTargetMappingFunc)
                    (HBA_HANDLE, HBA_FCPTARGETMAPPING *);
typedef HBA_STATUS  (* HBAGetFcpPersistentBindingFunc)
                    (HBA_HANDLE, HBA_FCPBINDING *);
typedef HBA_STATUS  (* HBAGetEventBufferFunc)
                    (HBA_HANDLE, HBA_EVENTINFO *, HBA_UINT32 *);
typedef HBA_STATUS  (* HBASetRNIDMgmtInfoFunc)
                    (HBA_HANDLE, HBA_MGMTINFO *);
typedef HBA_STATUS  (* HBAGetRNIDMgmtInfoFunc)
                    (HBA_HANDLE, HBA_MGMTINFO *);
```

```
typedef HBA_STATUS  (* HBASendRNIDFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_WWNTYPE, void *,HBA_UINT32 *);
typedef HBA_STATUS  (* HBASendScsiInquiryFunc)
                    (HBA_HANDLE, HBA_WWN,HBA_UINT64, HBA_UINT8,
                     HBA_UINT32, void *, HBA_UINT32, void *, HBA_UINT32);
typedef HBA_STATUS  (* HBASendReportLUNsFunc)
                    (HBA_HANDLE, HBA_WWN, void *, HBA_UINT32, void *,
                     HBA_UINT32);
typedef HBA_STATUS  (* HBASendReadCapacityFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_UINT64, void *, HBA_UINT32,
                     void *, HBA_UINT32);
```

The following are prototypes for the functions specified only in HBA API Phase 2 libraries:

```
typedef HBA_STATUS  (* HBAOpenAdapterByWWNFunc)
                    (HBA_HANDLE *, HBA_WWN);
typedef HBA_STATUS  (* HBAGetFcpTargetMappingV2Func)
                    (HBA_HANDLE, HBA_WWN, HBA_FCPTARGETMAPPING *);
typedef HBA_STATUS  (* HBASendCTPassThruV2Func)
                    (HBA_HANDLE, HBA_WWN, void *, HBA_UINT32, void *,
                     HBA_UINT32 *);
typedef void        (* HBARefreshAdapterConfigurationFunc) ();
typedef HBA_STATUS  (* HBAGetBindingCapabilityFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_BIND_CAPABILITY *);
typedef HBA_STATUS  (* HBAGetBindingSupportFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_BIND_CAPABILITY *);
typedef HBA_STATUS  (* HBASetBindingSupportFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_BIND_CAPABILITY);
typedef HBA_STATUS  (* HBASetPersistentBindingV2Func)
                    (HBA_HANDLE, HBA_WWN, const HBA_FCPBINDING2 *);
typedef HBA_STATUS  (* HBAGetPersistentBindingV2Func)
                    (HBA_HANDLE, HBA_WWN, HBA_FCPBINDING2 *);
typedef HBA_STATUS  (* HBARemovePersistentBindingFunc)
                    (HBA_HANDLE, HBA_WWN, const HBA_FCPBINDING2 *);
typedef HBA_STATUS  (* HBARemoveAllPersistentBindingsFunc)
                    (HBA_HANDLE, HBA_WWN);
typedef HBA_STATUS  (* HBASendRNIDV2Func)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_UINT32, HBA_UINT32, void *,
                     HBA_UINT32*);
typedef HBA_STATUS  (* HBAScsiInquiryV2Func)
                    (HBA_HANDLE,HBA_WWN,HBA_WWN, HBA_UINT64, HBA_UINT8,
                     HBA_UINT8, void *, HBA_UINT32 *, HBA_UINT8 *,
                     void *, HBA_UINT32 *);
typedef HBA_STATUS  (* HBAScsiReportLUNsV2Func)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, void *, HBA_UINT32 *,
                     HBA_UINT8 *, void *, HBA_UINT32 *);
typedef HBA_STATUS  (* HBAScsiReadCapacityV2Func)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_UINT64, void *,
                     HBA_UINT32 *, HBA_UINT8 *, void *, HBA_UINT32 *);
typedef HBA_UINT32  (* HBAGetVendorLibraryAttributesFunc)
                    (HBA_LIBRARYATTRIBUTES *);
typedef HBA_STATUS  (* HBARemoveCallbackFunc)
                    (HBA_CALLBACKHANDLE);
typedef HBA_STATUS  (* HBARegisterForAdapterAddEventsFunc)
                    (void (*)(void *, HBA_WWN, HBA_UINT32),
                    void *,HBA_CALLBACKHANDLE *);
typedef HBA_STATUS  (* HBARegisterForAdapterEventsFunc)
                    (void (*)(void *, HBA_WWN, HBA_UINT32),
                    void *,HBA_HANDLE, HBA_CALLBACKHANDLE *);
```

```
typedef HBA_STATUS  (* HBARegisterForAdapterPortEventsFunc)
                    (void (*)(void *, HBA_WWN, HBA_UINT32, HBA_UINT32),
                     void *, HBA_HANDLE, HBA_WWN, HBA_CALLBACKHANDLE *);
typedef HBA_STATUS  (* HBARegisterForAdapterPortStatEventsFunc)
                    (void (*)(void *, HBA_WWN, HBA_UINT32),
                    void *,HBA_HANDLE, HBA_WWN, HBA_PORTSTATISTICS,
                    HBA_UINT32, HBA_CALLBACKHANDLE *);
typedef HBA_STATUS  (* HBARegisterForTargetEventsFunc)
                    (void (*)(void *, HBA_WWN, HBA_WWN, HBA_UINT32),
                     void *, HBA_HANDLE, 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  (* HBASendRPLFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_UINT32,
                     HBA_UINT32, void *, HBA_UINT32 *);
typedef HBA_STATUS  (* HBASendRPSFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, HBA_UINT32, HBA_WWN,
                     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  (* HBAGetFC4StatisticsFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_UINT8,
                     HBA_FC4STATISTICS *);
typedef HBA_STATUS  (* HBAGetFCPStatisticsFunc)
                    (HBA_HANDLE, const HBA_SCSIID *,
                     HBA_FC4STATISTICS *);
typedef HBA_STATUS  (* HBASendRLSFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_WWN, void *, HBA_UINT32 *);
typedef HBA_STATUS  (* HBA_GetSBTargetMappingFunc)
                    (HBA_HANDLE, HBA_WWN, HBA_SBTARGETMAPPING *);
typedef HBA_STATUS  (* HBA_GetSBStatisticsFunc)
                    (HBA_HANDLE, const HBA_SBDEVID *, HBA_SBSTATISTICS *);
typedef HBA_STATUS  (* HBA_SBDskGetCapacityFunc)
                    (HBA_DEVICESELFDESC, HBA_SBDSKCAPACITY *)
```

The following structure is used to register a vendor-specific library that supports only Phase 1 functions. Although such libraries are supported, they are not compliant with this standard.

```
typedef struct HBA_EntryPoints {
      HBAGetVersionFunc                      GetVersionHandler;
      HBALoadLibraryFunc                     LoadLibraryHandler;
      HBAFreeLibraryFunc                     FreeLibraryHandler;
      HBAGetNumberOfAdaptersFunc             GetNumberOfAdaptersHandler;
      HBAGetAdapterNameFunc                  GetAdapterNameHandler;
      HBAOpenAdapterFunc                     OpenAdapterHandler;
      HBACloseAdapterFunc                    CloseAdapterHandler;
      HBAGetAdapterAttributesFunc            GetAdapterAttributesHandler;
      HBAGetAdapterPortAttributesFunc        GetAdapterPortAttributesHandler;
      HBAGetPortStatisticsFunc               GetPortStatisticsHandler;
      HBAGetDiscoveredPortAttributesFunc     GetDiscoveredPortAttributesHandler;
      HBAGetPortAttributesByWWNFunc          GetPortAttributesByWWNHandler;
      HBASendCTPassThruFunc                  SendCTPassThruHandler;
      HBARefreshInformationFunc              RefreshInformationHandler;
      HBAResetStatisticsFunc                 ResetStatisticsHandler;
      HBAGetFcpTargetMappingFunc             GetFcpTargetMappingHandler;
      HBAGetFcpPersistentBindingFunc         GetFcpPersistentBindingHandler;
      HBAGetEventBufferFunc                  GetEventBufferHandler;
      HBASetRNIDMgmtInfoFunc                 SetRNIDMgmtInfoHandler;
      HBAGetRNIDMgmtInfoFunc                 GetRNIDMgmtInfoHandler;
      HBASendRNIDFunc                        SendRNIDHandler;
      HBASendScsiInquiryFunc                 ScsiInquiryHandler;
      HBASendReportLUNsFunc                  ReportLUNsHandler;
      HBASendReadCapacityFunc                ReadCapacityHandler;
} HBA_ENTRYPOINTS, *PHBA_ENTRYPOINTS;
```

The following structure is used to register a compliant vendor-specific library on a system that does not support SB devices.

```
typedef struct HBA_EntryPointsV2 {
      HBAGetVersionFunc                      GetVersionHandler;
      HBALoadLibraryFunc                     LoadLibraryHandler;
      HBAFreeLibraryFunc                     FreeLibraryHandler;
      HBAGetNumberOfAdaptersFunc             GetNumberOfAdaptersHandler;
      HBAGetAdapterNameFunc                  GetAdapterNameHandler;
      HBAOpenAdapterFunc                     OpenAdapterHandler;
      HBACloseAdapterFunc                    CloseAdapterHandler;
      HBAGetAdapterAttributesFunc            GetAdapterAttributesHandler;
      HBAGetAdapterPortAttributesFunc        GetAdapterPortAttributesHandler;
      HBAGetPortStatisticsFunc               GetPortStatisticsHandler;
      HBAGetDiscoveredPortAttributesFunc     GetDiscoveredPortAttributesHandler;
      HBAGetPortAttributesByWWNFunc          GetPortAttributesByWWNHandler;
      HBASendCTPassThruFunc                  SendCTPassThruHandler;
      HBARefreshInformationFunc              RefreshInformationHandler;
      HBAResetStatisticsFunc                 ResetStatisticsHandler;
      HBAGetFcpTargetMappingFunc             GetFcpTargetMappingHandler;
      HBAGetFcpPersistentBindingFunc         GetFcpPersistentBindingHandler;
      HBAGetEventBufferFunc                  GetEventBufferHandler;
      HBASetRNIDMgmtInfoFunc                 SetRNIDMgmtInfoHandler;
      HBAGetRNIDMgmtInfoFunc                 GetRNIDMgmtInfoHandler;
      HBASendRNIDFunc                        SendRNIDHandler;
      HBASendScsiInquiryFunc                 ScsiInquiryHandler;
      HBASendReportLUNsFunc                  ReportLUNsHandler;
      HBASendReadCapacityFunc                ReadCapacityHandler;
      HBAOpenAdapterByWWNFunc                OpenAdapterByWWNHandler;
      HBAGetFcpTargetMappingV2Func           GetFcpTargetMappingV2Handler;
      HBASendCTPassThruV2Func                SendCTPassThruV2Handler;
      HBARefreshAdapterConfigurationFunc     RefreshAdapterConfigurationHandler;
      HBAGetBindingCapabilityFunc            GetBindingCapabilityHandler;
      HBAGetBindingSupportFunc               GetBindingSupportHandler;
      HBASetBindingSupportFunc               SetBindingSupportHandler;
      HBASetPersistentBindingV2Func          SetPersistentBindingV2Handler;
      HBAGetPersistentBindingV2Func          GetPersistentBindingV2Handler;
      HBARemovePersistentBindingFunc         RemovePersistentBindingHandler;
      HBARemoveAllPersistentBindingsFunc     RemoveAllPersistentBindingsHandler;
      HBASendRNIDV2Func                      SendRNIDV2Handler;
      HBAScsiInquiryV2Func                   ScsiInquiryV2Handler;
      HBAScsiReportLUNsV2Func                ScsiReportLUNsV2Handler;
      HBAScsiReadCapacityV2Func              ScsiReadCapacityV2Handler;
      HBAGetVendorLibraryAttributesFunc      GetVendorLibraryAttributesHandler;
      HBARemoveCallbackFunc                  RemoveCallbackHandler;
      HBARegisterForAdapterAddEventsFunc     RegisterForAdapterAddEventsHandler;
      HBARegisterForAdapterEventsFunc        RegisterForAdapterEventsHandler;
      HBARegisterForAdapterPortEventsFunc
                                   RegisterForAdapterPortEventsHandler;
      HBARegisterForAdapterPortStatEventsFunc
                                   RegisterForAdapterPortStatEventsHandler;
      HBARegisterForTargetEventsFunc         RegisterForTargetEventsHandler;
      HBARegisterForLinkEventsFunc           RegisterForLinkEventsHandler;
      HBASendRPLFunc                         SendRPLHandler;
      HBASendRPSFunc                         SendRPSHandler;
      HBASendSRLFunc                         SendSRLHandler;
      HBASendLIRRFunc                        SendLIRRHandler;
      HBAGetFC4StatisticsFunc                GetFC4StatisticsHandler;
      HBAGetFCPStatisticsFunc                GetFCPStatisticsHandler;
```

```
       HBASendRLSFunc                             SendRLSHandler;
} HBA_ENTRYPOINTSV2, *PHBA_ENTRYPOINTSV2;
```

The following structure is used to register a compliant vendor-specific library on a system that supports SB devices.

```
typedef struct HBA_EntryPointsV2 {
      HBAGetVersionFunc                       GetVersionHandler;
      HBALoadLibraryFunc                      LoadLibraryHandler;
      HBAFreeLibraryFunc                      FreeLibraryHandler;
      HBAGetNumberOfAdaptersFunc              GetNumberOfAdaptersHandler;
      HBAGetAdapterNameFunc                   GetAdapterNameHandler;
      HBAOpenAdapterFunc                      OpenAdapterHandler;
      HBACloseAdapterFunc                     CloseAdapterHandler;
      HBAGetAdapterAttributesFunc             GetAdapterAttributesHandler;
      HBAGetAdapterPortAttributesFunc         GetAdapterPortAttributesHandler;
      HBAGetPortStatisticsFunc                GetPortStatisticsHandler;
      HBAGetDiscoveredPortAttributesFunc      GetDiscoveredPortAttributesHandler;
      HBAGetPortAttributesByWWNFunc           GetPortAttributesByWWNHandler;
      HBASendCTPassThruFunc                   SendCTPassThruHandler;
      HBARefreshInformationFunc               RefreshInformationHandler;
      HBAResetStatisticsFunc                  ResetStatisticsHandler;
      HBAGetFcpTargetMappingFunc              GetFcpTargetMappingHandler;
      HBAGetFcpPersistentBindingFunc          GetFcpPersistentBindingHandler;
      HBAGetEventBufferFunc                   GetEventBufferHandler;
      HBASetRNIDMgmtInfoFunc                  SetRNIDMgmtInfoHandler;
      HBAGetRNIDMgmtInfoFunc                  GetRNIDMgmtInfoHandler;
      HBASendRNIDFunc                         SendRNIDHandler;
      HBASendScsiInquiryFunc                  ScsiInquiryHandler;
      HBASendReportLUNsFunc                   ReportLUNsHandler;
      HBASendReadCapacityFunc                 ReadCapacityHandler;
      HBAOpenAdapterByWWNFunc                 OpenAdapterByWWNHandler;
      HBAGetFcpTargetMappingV2Func            GetFcpTargetMappingV2Handler;
      HBASendCTPassThruV2Func                 SendCTPassThruV2Handler;
      HBARefreshAdapterConfigurationFunc      RefreshAdapterConfigurationHandler;
      HBAGetBindingCapabilityFunc             GetBindingCapabilityHandler;
      HBAGetBindingSupportFunc                GetBindingSupportHandler;
      HBASetBindingSupportFunc                SetBindingSupportHandler;
      HBASetPersistentBindingV2Func           SetPersistentBindingV2Handler;
      HBAGetPersistentBindingV2Func           GetPersistentBindingV2Handler;
      HBARemovePersistentBindingFunc          RemovePersistentBindingHandler;
      HBARemoveAllPersistentBindingsFunc      RemoveAllPersistentBindingsHandler;
      HBASendRNIDV2Func                       SendRNIDV2Handler;
      HBAScsiInquiryV2Func                    ScsiInquiryV2Handler;
      HBAScsiReportLUNsV2Func                 ScsiReportLUNsV2Handler;
      HBAScsiReadCapacityV2Func               ScsiReadCapacityV2Handler;
      HBAGetVendorLibraryAttributesFunc       GetVendorLibraryAttributesHandler;
      HBARemoveCallbackFunc                   RemoveCallbackHandler;
      HBARegisterForAdapterAddEventsFunc      RegisterForAdapterAddEventsHandler;
      HBARegisterForAdapterEventsFunc         RegisterForAdapterEventsHandler;
      HBARegisterForAdapterPortEventsFunc
                                  RegisterForAdapterPortEventsHandler;
      HBARegisterForAdapterPortStatEventsFunc
                                  RegisterForAdapterPortStatEventsHandler;
      HBARegisterForTargetEventsFunc          RegisterForTargetEventsHandler;
      HBARegisterForLinkEventsFunc            RegisterForLinkEventsHandler;
      HBASendRPLFunc                          SendRPLHandler;
      HBASendRPSFunc                          SendRPSHandler;
      HBASendSRLFunc                          SendSRLHandler;
      HBASendLIRRFunc                         SendLIRRHandler;
      HBAGetFC4StatisticsFunc                 GetFC4StatisticsHandler;
      HBAGetFCPStatisticsFunc                 GetFCPStatisticsHandler;
      HBASendRLSFunc                         SendRLSHandler;
      HBA_GetSBTargetMappingFunc             GetSBTargetMappingHandler;
      HBA_GetSBStatisticsFunc                GetSBStatisticsHandler;
      HBA_SBDskGetCapacityFunc               SBDskGetCapacityHandler;
} HBA_ENTRYPOINTSV2, *PHBA_ENTRYPOINTSV2;
```

```
/* This structure is defined here only for reference. */
/* It should be incorporated in code by including */
/* the appropriate system header file. */
struct tm {
      int          tm_sec;      /* seconds after the minute - [0,59] */
      int          tm_min;      /* minutes after the hour - [0,59] */
      int          tm_hour;     /* hours since midnight - [0,23] */
      int          tm_mday;     /* day of the month - [1,31] */
      int          tm_mon;      /* months since January - [0,11] */
      int          tm_year;     /* years since 1900 */
      int          tm_wday;     /* days since Sunday - [0,6] */
      int          tm_yday;     /* days since January 1 - [0,365] */
      int          tm_isdst;    /* daylight savings time flag */
};

typedef struct HBA_LibraryAttributes {
      HBA_BOOLEAN  final;
      char         LibPath[256];
      char         VName[256];
      char         VVersion[256];
      struct       tm build_date;
} HBA_LIBRARYATTRIBUTES, *PHBA_LIBRARYATTRIBUTES
```

6.11.2 Library Attribute Specifications

6.11.2.1 Compliance

Some library attributes shall be implemented as specified, and some shall either be implemented or given a value indicating they are not implemented (see annex A).

6.11.2.2 Final

If the library implements the final draft of the HBA API library specification version indicated by the value of HBA_GetVersion and similar functions, its final attribute shall be true. If the library implements a preliminary draft of the HBA API library specification version indicated by the function value, its final attribute shall be false.

6.11.2.3 LibPath

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

6.11.2.4 VName

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

6.11.2.5 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.

6.11.2.6 build_date

build_date shall be a standard tm structure containing the date/time at which the developing organization completed the code revision of the library being called. Zero values are acceptable for fields beyond the intended resolution of the developer. A compliant implementation may not provide correct values of the tm_wday, tm_yday, and tm_isdst fields.

7 Function Calls

7.1 Overview

Some HBA API functions shall be implemented as specified, and some shall either be implemented or return a value indicating they are not implemented (see annex A).

Table 3 is a directory to the functions specified by this standard.

Table 3 — Function Summary and Requirements

Function	Reference
Library Control Functions	
HBA_GetVersion	7.2.1
HBA_LoadLibrary	7.2.2
HBA_FreeLibrary	7.2.3
HBA_RegisterLibrary	7.2.4
HBA_RegisterLibraryV2	7.2.5
HBA_GetWrapperLibraryAttributes	7.2.6
HBA_GetVendorLibraryAttributes	7.2.7
HBA_GetNumberOfAdapters	7.2.8
HBA_RefreshInformation	7.2.9
HBA_RefreshAdapterConfiguration	7.2.10
HBA_ResetStatistics	7.2.11
HBA and Port Information Functions	
HBA_GetAdapterName	7.3.1
HBA_OpenAdapter	7.3.2
HBA_OpenAdapterByWWN	7.3.3
HBA_CloseAdapter	7.3.4
HBA_GetAdapterAttributes	7.3.5
HBA_GetAdapterPortAttributes	7.3.6
HBA_GetDiscoveredPortAttributes	7.3.7
HBA_GetPortAttributesByWWN	7.3.8
HBA_GetPortStatistics	7.3.9
HBA_GetFC4Statistics	7.3.10
FCP Information Functions	
HBA_GetBindingCapability	7.4.1
HBA_GetBindingSupport	7.4.2
HBA_SetBindingSupport	7.4.3
HBA_GetFcpTargetMapping	7.4.4

Table 3 — Function Summary and Requirements

Function	Reference
HBA_GetFcpTargetMappingV2	7.4.5
HBA_GetFcpPersistentBinding	7.4.6
HBA_GetPersistentBindingV2	7.4.7
HBA_SetPersistentBindingV2	7.4.8
HBA_RemovePersistentBinding	7.4.9
HBA_RemoveAllPersistentBindings	7.4.10
HBA_GetFCPStatistics	7.4.11
SCSI Information Functions	
HBA_SendScsiInquiry	7.5.1
HBA_ScsiInquiryV2	7.5.2
HBA_SendReportLUNs	7.5.3
HBA_ScsiReportLunsV2	7.5.4
HBA_SendReadCapacity	7.5.5
HBA_ScsiReadCapacityV2	7.5.6
SB Information Functions	
HBA_GetSBTargetMapping	7.6.1
HBA_GetSBStatistics	7.6.2
SB Disk Device Information Functions	
HBA_SBDskGetCapacity	7.7.1
Fabric Management Functions	
HBA_SendCTPassThru	7.8.1
HBA_SendCTPassThruV2	7.8.2
HBA_SetRNIDMgmtInfo	7.8.3
HBA_GetRNIDMgmtInfo	7.8.4
HBA_SendRNID	7.8.5
HBA_SendRNIDV2	7.8.6
HBA_SendRPL	7.8.7
HBA_SendRPS	7.8.8
HBA_SendSRL	7.8.9
HBA_SendLIRR	7.8.10

Table 3 — Function Summary and Requirements

Function	Reference
HBA_SendRLS	7.8.11
Event Handling Functions	
HBA_GetEventBuffer	7.9.2
HBA_RegisterForAdapterAddEvents	7.9.4
HBA_RegisterForAdapterEvents	7.9.5
HBA_RegisterForAdapterPortEvents	7.9.6
HBA_RegisterForAdapterPortStatEvents	7.9.7
HBA_RegisterForTargetEvents	7.9.8
HBA_RegisterForLinkEvents	7.9.9
HBA_RemoveCallback	7.9.10

7.2 Library Control Functions

7.2.1 HBA_GetVersion

7.2.1.1 Format

```
HBA_UINT32 HBA_GetVersion();
```

7.2.1.2 Description

The HBA_GetVersion function shall return the version of the HBA API specification with which the HBA API library is compatible.

An example usage of this function is provided in D.1

7.2.1.3 Arguments

None.

7.2.1.4 Return Values

function value shall indicate the version of the HBA API specification with which the library is compliant. The values shall be as specified in table 4.

Table 4 — Function Values for HBA_GetVersion

Function Value	Specification Version
1	Obsolete (see FC-MI)
2	This standard
any other	Reserved

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 perform common initialization, determine the configured HBA specific libraries, load the configured HBA specific libraries, load the HBA specific libraries’ function tables, and 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 have been loaded

The HBA_LoadLibrary function in an HBA specific library shall perform vendor specific initialization.

An example usage of this function is provided in D.2

7.2.2.3 Arguments

None.

7.2.2.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the library and all its configured HBA specific libraries loaded properly.
b) HBA_STATUS_ERROR_ALREADY_LOADED shall be returned to indicate that a library is already loaded.
c) HBA_STATUS_ERROR_INCOMPATIBLE shall be returned to indicate incompatibilities were detected among the wrapper library, its configured HBA specific libraries, and the drivers associated with the configured HBAs.
d) HBA_STATUS_ERROR may be returned to indicate any other problem with loading.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

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.

An example usage of this function is provided in D.3

7.2.3.3 Arguments

None.

7.2.3.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the library was able to free all resources.
b) HBA_STATUS_ERROR_NOT_LOADED shall be returned to indicate that there was no library currently loaded.
c) HBA_STATUS_ERROR may be returned to indicate any problem with freeing resources.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

7.2.4 HBA_RegisterLibrary

7.2.4.1 Format

```
HBA_STATUS HBA_RegisterLibrary(
      HBA_ENTRYPOINTS *pHBAInfo
);
```

7.2.4.2 Description

The HBA_RegisterLibrary function shall register the Phase I functionality of an HBA specific library with the wrapper library. This shall be implemented only by an HBA specific library and called by the wrapper library (i.e., clients of the HBA API specified by this standard shall not refer to this function directly). HBA specific libraries compliant with this standard shall support this function to preserve compatibility with Phase I wrapper libraries.

An example implementation of this function is provided in D.4

7.2.4.3 Arguments

pHBAInfo shall be a pointer to a structure in which the entry addresses of the vendor specific implementations of the Phase I library functions may be returned.

7.2.4.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate function pointers have been returned.
b) HBA_STATUS_ERROR may be returned to indicate any problem with returning pointers.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pHBAInfo shall be unchanged. The structure to which it points shall contain the entry addresses of the vendor specific implementations of each Phase I library function.

7.2.5 HBA_RegisterLibraryV2

7.2.5.1 Format

```
HBA_STATUS HBA_RegisterLibraryV2(
      HBA_ENTRYPOINTSV2 *pHBAInfo
);
```

7.2.5.2 Description

The HBA_RegisterLibraryV2 function shall register the Phase II functionality of an HBA specific library with the wrapper library. This shall be implemented only by an HBA specific library and called by the wrapper library (i.e., clients of the HBA API specified by this standard shall not refer to this function directly). HBA specific libraries compliant with this standard shall support both this function (HBA_RegisterLibraryV2) and HBA_RegisterLibrary.

7.2.5.3 Arguments

pHBAInfo shall be a pointer to a structure in which the entry addresses of the vendor specific implementations of all library functions may be returned.

7.2.5.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate function pointers have been returned.
b) HBA_STATUS_ERROR may be returned to indicate any problem with returning pointers.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pHBAInfo shall be unchanged. The structure to which it points shall contain the entry addresses of the vendor specific implementations of all library functions.

7.2.6 HBA_GetWrapperLibraryAttributes

7.2.6.1 Format

```
HBA_UINT32 HBA_GetWrapperLibraryAttributes(
      HBA_LIBRARYATTRIBUTES *attributes
);
```

7.2.6.2 Description

The HBA_GetWrapperLibraryAttributes function shall return details about the implementation of the wrapper library in which the call is implemented. Its intended uses include to allow software to determine whether a compatible library is installed and to allow installation software to describe to an operator a library about to be replaced.

In an HBA API library with OS specific structure, the HBA_GetWrapperLibraryAttributes function returns information about the OS specific software that presents the API.

7.2.6.3 Arguments

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

function value shall indicate the version of the HBA API specification with which the library is compliant. The values shall be as specified in table 5.

Table 5 — Function Values for HBA_GetWrapperLibraryAttributes

Function Value	Specification Version
1	Obsolete (see FC-MI)
2	This standard
any other	Reserved

attributes shall be unchanged. The structure to which it points shall contain the attributes of the library implementing the call. If it is not practical to determine the LibPath attribute, a null string may be returned for that attribute.

7.2.7 HBA_GetVendorLibraryAttributes

7.2.7.1 Format

```
HBA_UINT32 HBA_GetVendorLibraryAttributes(
       HBA_UINT32 adapter_index,
       HBA_LIBRARYATTRIBUTES *attributes
);
```

7.2.7.2 Description

The HBA_GetVendorLibraryAttributes function shall return details about the implementation of the HBA specific library associated with the specified HBA. Its intended uses include to allow software, including a wrapper library, to determine whether a compatible library is installed and to allow installation software to describe to an operator a library about to be replaced.

In an HBA API library with OS specific structure, the HBA_GetVendorLibraryAttributes function returns information about the OS specific software that presents the API. This shall be the same as the information returned by HBA_GetWrapperLibraryAttributes.

7.2.7.3 Arguments

adapter_index shall be an index to an HBA in the range of the return value of HBA_GetNumberOfAdapters. The version details shall be returned for the HBA specific library that interfaces to the indexed HBA.

In an HBA API library with OS specific structure, the HBA_GetVendorLibraryAttributes function returns information about the OS specific software that presents the API regardless of the HBA that is indexed.

NOTE 10 An index is used rather than a name or handle so that this service may be called without opening an HBA.

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.

attributes shall be a pointer to a structure in which the attributes of the library implementing the call may be returned.

7.2.7.4 Return Values

function value shall indicate the version of the HBA API specification with which the library is compliant. The values shall be as specified in table 6.

Table 6 — Function Values for HBA_GetVendorLibraryAttributes

Function Value	Specification Version
1	Obsolete (see FC-MI)
2	This standard
any other	Reserved

attributes shall be unchanged. The structure to which it points shall contain the attributes of the specified HBA specific library. If it is not practical to determine the LibPath attribute, a null string may be returned for that attribute.

7.2.8 HBA_GetNumberOfAdapters

7.2.8.1 Format

```
HBA_UINT32 HBA_GetNumberOfAdapters();
```

7.2.8.2 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.

An example usage of this function is provided in D.5

7.2.8.3 Arguments

None.

7.2.8.4 Return Values

function value shall be the number of adapters supported by this library. If no adapters are supported, the library shall return 0.

7.2.9 HBA_RefreshInformation

7.2.9.1 Format

```
void HBA_RefreshInformation(
      HBA_HANDLE handle
);
```

7.2.9.2 Description

The HBA_RefreshInformation function shall cause information saved by the HBA API about the specified HBA to be made current.

An HBA API shall return a HBA_STATUS_ERROR_STALE_DATA error to any call identifying an HBA end port or discovered FC_Port by an index into an implied internal table (i.e., HBA_GetAdapterPortAttributes, HBA_GetDiscoveredPortAttributes) if the assignment of indexes in the implied table has changed since the application last called HBA_RefreshInformation. If an HBA API returns HBA_STATUS_ERROR_STALE_DATA to an application on a call referring to an HBA, the HBA API shall continue to return HBA_STATUS_ERROR_STALE_DATA to that application for all calls referring to the same HBA until that application calls HBA_RefreshInformation for the HBA, indicating to the HBA API that the application is prepared to deal with the changes.

An example usage of this function is provided in D.6

7.2.9.3 Arguments

handle shall be a handle to an open HBA.

7.2.9.4 Return Values

None.

7.2.10 HBA_RefreshAdapterConfiguration

7.2.10.1 Format

```
void HBA_RefreshAdapterConfiguration(
);
```

7.2.10.2 Description

The HBA_RefreshAdapterConfiguration function shall cause information saved by the HBA API about the HBAs present in the system to be made current.

NOTE 11 HBA_RefreshAdapterConfiguration is intended to support dynamic HBA reconfiguration in HBA API library implementations that comply with strictly static implicit tables by explicitly provoking the library to discover and assign HBA indexes and names to newly installed HBAs. This relieves the library of the need to poll for new adapters.

The HBA_RefreshAdapterConfiguration function shall not interfere with any established relationships between software and adapters that have not been reconfigured. Thus, these relationships shall survive an invocation of HBA_RefreshAdapterConfiguration:

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

NOTE 12 These rules imply that in systems that allow dynamic HBA reconfiguration, indexes assigned to removed adapters may be interspersed with indexes to installed adapters. In systems that contain adapters from multiple vendors and allow dynamic HBA reconfiguration, it may not be possible for the wrapper library to assign contiguous HBA indexes to adapters from the same vendor.

A driver may have the capability of recognizing (or being told of) a functionally equivalent replacement of a removed HBA that may have different identifying information. In this case, the replacement HBA should occupy the HBA index and name of the removed HBA after HBA_RefreshAdapterConfiguration is called. If an HBA API chooses to implement a strictly static table model, before HBA_RefreshAdapterConfiguration is called but after the replacement is inserted, HBA_STATUS_ERROR_STALE_DATA or HBA_STATUS_ERROR_UNAVAILABLE shall be returned on any reference to identifiers of the removed HBA.

7.2.10.3 Arguments

None.

7.2.10.4 Return Values

None.

7.2.11 HBA_ResetStatistics

7.2.11.1 Format

```
void HBA_ResetStatistics(
       HBA_HANDLE handle,
       HBA_UINT32 portindex
);
```

7.2.11.2 Description

The HBA_ResetStatistics function is obsolete. It is retained for compatibility with very early implementations of HBA API clients. It shall have no effect and return no value.

7.3 HBA and Port Information Functions

7.3.1 HBA_GetAdapterName

7.3.1.1 Format

```
HBA_STATUS HBA_GetAdapterName(
       UINT32 adapterindex,
       char *pAdaptername
);
```

7.3.1.2 Description

The HBA_GetAdapterName function shall return the text string that identifies this HBA. The text string may be used to open the HBA with the library as well as for a human-readable identification of an HBA instance. The name shall be derived from the reversed domain name of the manufacturer.

An example usage of this function is provided in D.7

7.3.1.3 Arguments

adapterindex shall be the index of the HBA for which the name is to be returned.

pAdaptername shall be a pointer to space in which the HBA name may be returned as an ASCII string.

7.3.1.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate successful completion.
b) HBA_STATUS_ERROR_ILLEGAL_INDEX if there is no adapter with the given index.
c) HBA_STATUS_ERROR may be returned to indicate any problem with returning attributes.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pAdaptername shall be unchanged. The buffer to which it points shall contain an ASCII string in the form:

mfgdomain-model-adapterindex

where mfgdomain shall be the reverse form of a domain name owned by the manufacturer of the HBA; model shall be a vendor specific identifier of the HBA product model; and adapterindex shall be a decimal number distinguishing multiple instances of the same model registered with the HBA API

Example:

```
com.HotBiscuitsAdapters-HBA1040A-1
```

7.3.2 HBA_OpenAdapter

7.3.2.1 Format

```
HBA_HANDLE HBA_OpenAdapter(
      char *pAdaptername
);
```

7.3.2.2 Description

The HBA_OpenAdapter function shall open a named HBA. By opening an HBA, an upper level application is ensuring that all access to an HBA_HANDLE between an open and a close shall be to the same HBA. An HBA_OpenAdapter may not imply a driver open; that is vendor implementation dependent.

An example usage of this function is provided in D.8

7.3.2.3 Arguments

pAdaptername shall be an adapter name returned by HBA_GetAdapterName.

7.3.2.4 Return Values

function value shall be a valid HBA_HANDLE on success, or zero on failure.

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 that is opened.

7.3.3 HBA_OpenAdapterByWWN

7.3.3.1 Format

```
HBA_STATUS HBA_OpenAdapterByWWN(
       HBA_HANDLE *pHandle,
       HBA_WWN wwn
);
```

7.3.3.2 Description

The HBA_OpenAdapterByWWN function shall attempt to open a handle to the HBA that contains a Node_Name or N_Port_Name matching the wwn argument. The specified WWN shall match the HBA's Node_Name or N_Port_Name. Discovered end ports (remote end ports) shall NOT be checked for a match.

7.3.3.3 Arguments

pHandle shall be a pointer to a handle. The value at entry is irrelevant.

wwn shall be a WWN to match the Node_Name or N_Port_Name of the HBA to open.

7.3.3.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the handle contains a valid HBA handle.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate no HBA has a Node_Name or N_Port_Name that matches wwn.
c) HBA_STATUS_ERROR_AMBIGUOUS_WWN shall be returned to indicate multiple adapters have a matching WWN. This may occur if the Node_Names of multiple adapters are identical.
d) HBA_STATUS_ERROR may be returned to indicate any other problem with opening the adapter.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pHandle shall be unchanged. If the open succeeds, the value to which it points shall be a handle to the requested HBA. On failure, the value is undefined.

7.3.4 HBA_CloseAdapter

7.3.4.1 Format

```
void HBA_CloseAdapter(
       HBA_HANDLE handle
);
```

7.3.4.2 Description

Function HBA_CloseAdapter shall close an open HBA.

An example usage of this function is provided in D.9

7.3.4.3 Arguments

handle shall be a HBA_HANDLE to an opened HBA that is to be closed.

7.3.4.4 Return Values

None.

7.3.5 HBA_GetAdapterAttributes

7.3.5.1 Format

```
HBA_STATUS HBA_GetAdapterAttributes(
       HBA_HANDLE handle,
       HBA_ADAPTERATTRIBUTES *pAdapterattributes
);
```

7.3.5.2 Description

The HBA_GetAdapterAttributes function shall retrieve the attributes for an HBA.

An example usage of this function is provided in D.10

7.3.5.3 Arguments

handle shall be a HBA_HANDLE to an opened HBA for which attributes are requested.

pAdapterattributes shall be a pointer to a structure in which attributes for the HBA may be returned

7.3.5.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the attributes of an HBA have been returned.
b) HBA_STATUS_ERROR may be returned to indicate any problem with getting attributes.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pAdapterattributes shall be unchanged. The structure to which it points shall contain attributes for the HBA. HBA attributes may include:

```
Manufacturer
SerialNumber
Model
ModelDescription
NodeWWN
NodeSymbolicName
HardwareVersion
```

```
        DriverVersion
        OptionROMVersion
        FirmwareVersion
        VendorSpecificID
        NumberOfPorts
        DriverName
```

7.3.6 HBA_GetAdapterPortAttributes

7.3.6.1 Format

```
HBA_STATUS HBA_GetAdapterPortAttributes(
        HBA_HANDLE handle,
        HBA_UINT32 portindex,
        HBA_PORTATTRIBUTES *pPortattributes
);
```

7.3.6.2 Description

Function HBA_GetAdapterPortAttributes shall retrieve the attributes for a specified end port on an HBA.

An example usage of this function is provided in D.11

7.3.6.3 Arguments

handle shall be an HBA_HANDLE to an opened HBA.

portindex shall be the index within the specified HBA of the end port to query.

pPortattributes shall be a pointer to a structure in which attributes for the end port may be returned

7.3.6.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the attributes of an end port on an HBA have been returned.
b) HBA_STATUS_ERROR_ILLEGAL_INDEX if there is no end port with the given index.
c) HBA_STATUS_ERROR may be returned to indicate any problem with getting attributes.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pPortattributes shall be unchanged. The structure to which it points shall contain attributes for the end port. Port attributes may include:

```
        NodeWWN
        PortWWN
        PortFcId
        PortType
        PortState
        PortSupportedClassofService
        PortSupportedFc4Types
        PortActiveFc4Types
        OSDeviceName
```

```
       PortSpeed
       NumberofDiscoveredPorts
       PortSymbolicName
       PortSupportedSpeed
       PortMaxFrameSize
       FabricName
```

7.3.7 HBA_GetDiscoveredPortAttributes

7.3.7.1 Format

```
HBA_STATUS HBA_GetDiscoveredPortAttributes(
       HBA_HANDLE handle,
       HBA_UINT32 portindex,
       HBA_UINT32 discoveredportindex,
       HBA_PORTATTRIBUTES *pPortattributes
);
```

7.3.7.2 Description

The HBA_GetDiscoveredPortAttributes function shall retrieve the attributes for a specified FC_Port discovered in the network.

An example usage of this function is provided in D.12

7.3.7.3 Arguments

handle shall be a HBA_HANDLE to an opened HBA containing the local end port for which a discovered FC_Port is to be queried.

portindex shall be the index on the HBA of the local end port through which to query the discovered FC_Port.

discoveredportindex shall be the index of the discovered FC_Port to query.

pPortattributes shall be a pointer to a structure in which attributes for the discovered FC_Port may be returned

7.3.7.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the attributes of an FC_Port visible to the specified local end port have been returned.
b) HBA_STATUS_ERROR_ILLEGAL_INDEX if there is no end port with the given index.
c) HBA_STATUS_ERROR may be returned to indicate any problem with returning attributes.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pPortattributes shall be unchanged. The structure to which it points shall contain attributes for the discovered FC_Port. Attributes for a discovered FC_Port may include:

```
       NodeWWN
       PortWWN
       PortFcId
       PortType
       PortState
       PortSupportedClassofService
       PortSupportedFc4Types
       PortActiveFc4Types
       OSDeviceName
       PortSpeed
       PortSymbolicName
       PortSupportedSpeed
       PortMaxFrameSize
       FabricName
```

In the case of HBA_GetDiscoveredPortAttributes NumberOfDiscoveredPorts shall be 0.

7.3.8 HBA_GetPortAttributesByWWN

7.3.8.1 Format

```
HBA_STATUS HBA_GetPortAttributesByWWN(
       HBA_HANDLE handle,
       HBA_WWN PortWWN,
       HBA_PORTATTRIBUTES *pPortattributes
);
```

7.3.8.2 Description

The HBA_GetPortAttributesByWWN function shall retrieve the attributes for a local HBA end port or discovered FC_Port specified by Port Name.

7.3.8.3 Arguments

handle shall be a HBA_HANDLE to an opened HBA.

PortWWN shall be the Port Name of the FC_Port to query.

pPortattributes shall be a pointer to a structure in which attributes for the FC_Port may be returned

7.3.8.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the attributes of a FC_Port with the specified Port Name have been returned.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle is not able to access a local end port or discovered FC_Port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR may be returned to indicate any problem with returning attributes.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pPortattributes shall be unchanged. The structure to which it points shall contain attributes for the FC_Port. FC_Port attributes are as described in HBA_GetAdapterPortAttributes (see 7.3.6) for local HBA end ports and in HBA_GetDiscoveredPortAttributes (see 7.3.7) for discovered FC_Ports:

7.3.9 HBA_GetPortStatistics

7.3.9.1 Format

```
HBA_STATUS HBA_GetPortStatistics(
       HBA_HANDLE handle,
       HBA_UINT32 portindex,
       HBA_PORTSTATISTICS *pPortstatistics
);
```

7.3.9.2 Description

The HBA_GetPortStatistics function shall retrieve the statistics for a specified end port on an HBA. The exact meaning of events being counted for each statistic is vendor specific. LinkFailureCount, LossOfSyncCount, LossOfSignalCount, PrimitiveSeqProtocolErrCount, InvalidTxWordCount, and InvalidCRCCount shall be the values that are maintained by the end port in its Link Error Status Block (see FC-FS).

An example usage of this function is provided in D.13

7.3.9.3 Arguments

handle shall be a HBA_HANDLE to an opened HBA for which end port statistics are to be returned.

portindex Shall be the index of the end port to query.

pPortstatistics shall be a pointer to a structure in which statistics for the end port may be returned.

7.3.9.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the statistics for the specified end port have been returned.
b) HBA_STATUS_ERROR_ILLEGAL_INDEX if there is no end port with the given index.
c) HBA_STATUS_ERROR may be returned to indicate any problem with returning statistics.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pPortstatistics shall be unchanged. The structure to which it points shall contain statistics for the end port. End port statistics may include:

```
SecondsSinceLastReset
TxFrames
TxWords
RxFrames
RxWords
LIPCount
NOSCount
ErrorFrames
DumpedFrames
LinkFailureCount
LossOfSyncCount
LossOfSignalCount
PrimitiveSeqProtocolErrCount
```

```
       InvalidTxWordCount
       InvalidCRCCount
```

7.3.10 HBA_GetFC4Statistics

7.3.10.1 Format

```
HBA_STATUS HBA_GetFC4Statistics(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_UINT8 FC4type,
       HBA_FC4STATISTICS *statistics
);
```

7.3.10.2 Description

The HBA_GetFC4Statistics function shall return traffic statistics for a specific FC-4 protocol via a specific local HBA and local end port.

NOTE 13 Basic Link Service, Extended Link Service, and CT each have specific Data Structure TYPE values, so their traffic may be requested.

7.3.10.3 Arguments

handle shall be a handle to an open HBA containing the end port for which to return FC-4 statistics.

hbaPortWWN shall be the Port Name of the local HBA end port for which to return FC-4 statistics.

FC4type shall be the Data Structure TYPE assigned by FC-FS to the FC-4 protocol for which FC-4 statistics are requested.

statistics shall be a pointer to an FC-4 Statistics structure in which the statistics for the specified FC-4 protocol may be returned.

7.3.10.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the statistics for the specified FC-4 and end port have been returned.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR_UNSUPPORTED_FC4 shall be returned to indicate the specified HBA end port does not support the specified FC-4 protocol.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

statistics shall be unchanged. The structure to which it points shall contain the statistics for the specified FC-4 protocol.

7.4 FCP Information Functions

7.4.1 HBA_GetBindingCapability

7.4.1.1 Format

```
HBA_STATUS HBA_GetBindingCapability(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_BIND_CAPABILITY *pFlags
);
```

7.4.1.2 Description

The HBA_GetBindingCapability function shall return the binding capabilities implemented for a specified HBA end port.

7.4.1.3 Arguments

handle shall be a handle to an open HBA containing the end port for which to return implemented persistent binding capabilities.

hbaPortWWN shall be the Port Name of the local HBA end port for which to return implemented persistent binding capabilities.

pFlags shall point to an HBA_BIND_CAPABILITY structure in which to return implemented persistent binding capabilities.

7.4.1.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the persistent binding capabilities implemented by the specified end port have been returned.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA referenced by handle does not support persistent binding.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pFlags shall be unchanged. In the structure to which it points, the implemented persistent binding capabilities shall be indicated.

7.4.2 HBA_GetBindingSupport

7.4.2.1 Format

```
HBA_STATUS HBA_GetBindingSupport(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_BIND_CAPABILITY *pFlags
);
```

7.4.2.2 Description

The HBA_GetBindingSupport function shall return the binding capabilities currently enabled for a specified HBA end port.

7.4.2.3 Arguments

handle shall be a handle to an open HBA containing the end port for which to return currently enabled persistent binding capabilities.

hbaPortWWN shall be the Port Name of the local HBA end port for which to return currently enabled persistent binding capabilities.

pFlags shall point to an HBA_BIND_CAPABILITY structure in which to return currently enabled persistent binding capabilities.

7.4.2.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the persistent binding capabilities currently enabled by the specified end port have been returned.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA referenced by handle does not support persistent binding.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pFlags shall be unchanged. In the structure to which it points, the currently enabled persistent binding capabilities shall be indicated.

7.4.3 HBA_SetBindingSupport

7.4.3.1 Format

```
HBA_STATUS HBA_SetBindingSupport(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_BIND_CAPABILITY flags
);
```

7.4.3.2 Description

The HBA_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 HBA_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 in disabling other capabilities, though imaginative developers may.

7.4.3.3 Arguments

handle shall be a handle to an open HBA containing the end port for which to set the currently enabled persistent binding capabilities.

hbaPortWWN shall be the Port Name of the local HBA end port for which to set the currently enabled persistent binding capabilities.

flags shall point to an HBA_BIND_CAPABILITY structure indicating persistent binding capabilities to enable.

7.4.3.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate persistent binding capabilities have been enabled by the specified end port as requested.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA referenced by handle does not support persistent binding.
d) HBA_ERROR_INCAPABLE shall be returned to indicate the flags include a flag for a capability not implemented for the referenced end port.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

7.4.4 HBA_GetFcpTargetMapping

7.4.4.1 Format

```
HBA_STATUS HBA_GetFcpTargetMapping(
       HBA_HANDLE handle,
       HBA_FCPTARGETMAPPING *pMapping
);
```

7.4.4.2 Description

The HBA_GetFcpTargetMapping function shall return the mapping between OS identification of SCSI targets or logical units and FCP identification of targets or logical units offered by the specified HBA at the time the function call is processed (see SAM-3, FCP-2, and relevant OS documentation). Space in the pMapping structure permitting, one mapping entry shall be returned for each FCP logical unit represented in the OS and one mapping entry shall be returned for each FCP target that is represented in the OS but for which no logical units are represented in the OS. No target mapping entries shall be returned to represent FCP objects that are not represented in the OS (i.e., are unmapped).

7.4.4.3 Arguments

handle shall be a handle to an open HBA for which to retrieve the mapping.

pMapping shall be a pointer to an HBA_FCPTARGETMAPPING structure. The size of this structure shall be limited by the NumberOfEntries value within the structure at function call.

7.4.4.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate all mapping entries have been returned for the specified end port.
b) HBA_STATUS_ERROR_MORE_DATA shall be returned to indicate more space in the buffer is required to contain mapping information.
c) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pMapping shall be unchanged. The structure to which it points shall contain mapping information from OS identifications of SCSI logical units to FCP identifications of logical units for this HBA (see SAM-3, FCP-2, and relevant OS documentation). The number of entries returned 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 structure shall be the total number of mappings the HBA has established even when the function returns an error because the buffer is too small to return all of the established mappings. An upper level 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.

7.4.5 HBA_GetFcpTargetMappingV2

7.4.5.1 Format

```
HBA_STATUS HBA_GetFcpTargetMappingV2(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_FCPTARGETMAPPINGV2 *pMapping
);
```

7.4.5.2 Description

The HBA_GetFcpTargetMappingV2 function shall return the mapping between OS identification of SCSI targets or logical units and FCP identification of targets or logical units offered by the specified HBA end port at the time the function call is processed. Space in the pMapping structure permitting, one mapping entry shall be returned for each FCP logical unit represented in the OS and one mapping entry shall be returned for each FCP target that is represented in the OS but for which no logical units are represented in the OS. No target mapping entries shall be returned to represent FCP 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-3, FCP-2, and relevant OS documentation). If the VPD Page 83 information for a logical unit provides more than one LUID, the one returned shall be the type 3 (FC Name_Identifier) LUID with the smallest identifier value if any LUID of type 3 is provided; otherwise, the type 2 (IEEE EUI-64) LUID with the smallest identifier value if any LUID of type 2 is provided; otherwise, the type 1 (T10 vendor identification) LUID with the smallest identifier value if any LUID of type 1 is provided; otherwise, the type 0 (vendor specific) LUID with the smallest identifier value. If the logical unit provides no LUID, the value of the first four bytes of the LUID field shall be zero.

7.4.5.3 Arguments

handle shall be a handle to an open HBA containing the end port for which target mappings are requested.

hbaPortWWN shall be the Port Name of the local HBA end port for which target mappings are requested.

pMapping Pointer to an HBA_FCPTARGETMAPPINGV2 structure. The size of this structure shall be limited by the NumberOfEntries value within the structure.

7.4.5.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate all mapping entries have been returned for the specified end port.
b) HBA_STATUS_ERROR_MORE_DATA shall be returned to indicate more space in the buffer is required to contain mapping information.
c) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
d) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA referenced by handle does not support target mapping.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pMapping shall be unchanged. The structure to which it points shall contain mapping information from OS identifications of SCSI logical units to FCP identifications of logical units for the specified local HBA end port (see SAM-3, FCP-2, 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 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 upper level 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.

7.4.6 HBA_GetFcpPersistentBinding

7.4.6.1 Format

```
HBA_STATUS HBA_GetFcpPersistentBinding(
       HBA_HANDLE handle,
       HBA_FCPBINDING *pBinding
);
```

7.4.6.2 Description

The HBA_GetFcpPersistentBinding function shall return the persistent bindings that direct the HBA in establishing mappings from OS identifications of SCSI logical units to FCP identifications of logical units (see SAM-3, FCP-2, and relevant OS documentation).

7.4.6.3 Arguments

handle shall be a handle to an open HBA.

pBinding shall be a pointer to a HBA_FCPBINDING structure. The size of this structure shall be limited by the NumberOfEntries value within the structure.

7.4.6.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate all binding entries have been returned for the specified end port.
b) HBA_STATUS_ERROR_MORE_DATA shall be returned to indicate more space in the buffer is required to contain binding information.
c) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
d) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA referenced by handle does not support persistent binding.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pBinding shall be unchanged. The structure to which it points shall contain binding information from OS identifications of SCSI logical units to FCP identifications of logical units for the specified HBA (see SAM-3, FCP-2, 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 set of bindings. The value of the NumberOfEntries field of the returned structure shall be the total number of persistent bindings the HBA has established even when the function returns an error because the buffer is too small to return all of the established bindings. An upper level 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.

7.4.7 HBA_GetPersistentBindingV2

7.4.7.1 Format

```
HBA_STATUS HBA_GetPersistentBindingV2(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_FCPBINDING2 *binding
);
```

7.4.7.2 Description

The HBA_GetFcpPersistentBindingV2 function shall return persistent bindings between an FCP target and a SCSI ID for a specified HBA end port. The binding information may include bindings to Logical Unit Unique Device Identifiers.

7.4.7.3 Arguments

handle shall be a handle to an open HBA containing the end port for which to return persistent binding.

hbaPortWWN shall be the Port Name of the local HBA end port for which to return persistent binding.

binding shall be a pointer to an HBA_FCPBINDING2 structure. The NumberOfEntries field in the structure shall limit the number of entries that shall be returned.

7.4.7.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate all binding entries have been returned for the specified end port.
b) HBA_STATUS_ERROR_MORE_DATA shall be returned to indicate more space in the buffer is required to contain binding information.
c) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
d) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA referenced by handle does not support persistent binding.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

binding shall be unchanged. The structure to which it points shall contain binding information from OS identifications of SCSI logical units to FCP and LUID identifications of logical units for the specified HBA end port (see SAM-3, FCP-2, 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 set of bindings. The NumberOfEntries field shall contain the total number of bindings established by the end port. An application may either call HBA_GetPersistentBindingV2 with NumberOfEntries set to 0 to retrieve the number of entries available, or allocate a sufficiently large buffer to retrieve entries at first call. The Status field of each HBA_FCPBINDINGENTRY2 substructure shall be zero.

7.4.8 HBA_SetPersistentBindingV2

7.4.8.1 Format

```
HBA_STATUS HBA_SetPersistentBindingV2(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_FCPBINDING2 *binding
);
```

7.4.8.2 Description

The HBA_SetPersistentBindingV2 function shall set additional persistent bindings between SCSI IDs and FCP targets for the specified HBA end port. It shall accept extended bindings to Logical Unit Unique Device Identifiers. Bindings already in effect shall remain in effect. A requested binding to the same local OS SCSI ID as a binding that is already in effect shall be errored. 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.

7.4.8.3 Arguments

handle shall be a handle to an open HBA containing the end port for which to set persistent binding.

hbaPortWWN shall be the Port Name of the local HBA end port for which to set persistent binding.

binding shall be a pointer to an HBA_FCPBINDING2 structure. The NumberOfEntries field in the structure shall determine the number of requested entries in the structure.

7.4.8.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the requested persistent bindings have been set for the specified end port.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA referenced by handle does not support persistent binding.
d) HBA_STATUS_ERROR_LOCAL_SCSIID_BOUND shall be returned to indicate a persistent binding set request included a local SCSI ID that was already bound.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

binding shall be unchanged. In the structure to which it points, the success or failure of setting the binding requested by each HBA_FCPBINDINGENTRY2 substructure shall be indicated by setting the value of the Status field in the substructure to a value defined in 6.2

7.4.9 HBA_RemovePersistentBinding

7.4.9.1 Format

```
HBA_STATUS HBA_RemovePersistentBinding(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_FCPBINDING2 *binding
);
```

7.4.9.2 Description

The HBA_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 FCP ID 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.

7.4.9.3 Arguments

handle shall be a handle to an open HBA containing the end port from which to remove persistent bindings.

hbaPortWWN shall be the Port Name of the local HBA end port for which to remove persistent bindings.

binding shall be a pointer to a HBA_FCPBINDING2 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.

7.4.9.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the requested persistent bindings have been removed for the specified end port.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA referenced by handle does not support persistent binding.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

binding shall be unchanged. In the structure to which it points, the success or failure of removing the binding requested by each HBA_FCPBINDINGENTRY2 substructure shall be indicated by setting the value of the Status field in the substructure to a value defined in 6.2

7.4.10 HBA_RemoveAllPersistentBindings

7.4.10.1 Format

```
HBA_STATUS HBA_RemoveAllPersistentBindings(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN
);
```

7.4.10.2 Description

The HBA_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.

7.4.10.3 Arguments

handle shall be a handle to an open HBA containing the end port from which to remove all persistent bindings.

hbaPortWWN shall be the Port Name of the local HBA end port from which to remove all persistent bindings.

7.4.10.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate all persistent bindings have been removed for the specified end port.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.

c) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA referenced by handle does not support persistent binding.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

7.4.11 HBA_GetFCPStatistics

7.4.11.1 Format

```
HBA_STATUS HBA_GetFCPStatistics(
      HBA_HANDLE handle,
      const HBA_SCSIID *lunit,
      HBA_FC4STATISTICS *statistics
);
```

7.4.11.2 Description

The HBA_GetFCPStatistics function shall return traffic statistics for a specific OS SCSI logical unit provided via the FCP protocol on a specific local HBA.

7.4.11.3 Arguments

handle shall be a handle to an open HBA for which to return FCP-2 statistics.

lunit shall be a pointer to a structure specifying the OS SCSI logical unit for which FCP-2 statistics are requested

statistics shall be a pointer to a FC-4 Statistics structure in which the FCP-2 statistics for the specified logical unit may be returned.

7.4.11.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate FCP-2 statistics have been returned for the specified HBA.
b) HBA_STATUS_ERROR_INVALID_LUN shall be returned to indicate the HBA referenced by handle does not support the logical unit referenced by lunit.
c) HBA_STATUS_ERROR_UNSUPPORTED_FC4 shall be returned to indicate the specified HBA end port does not support FCP-2.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

statistics shall be unchanged. The structure to which it points shall contain the FCP-2 statistics for the specified HBA and logical unit.

7.5 SCSI Information Functions

7.5.1 HBA_SendScsiInquiry

7.5.1.1 Format

```
HBA_STATUS HBA_SendScsiInquiry(
       HBA_HANDLE handle,
       HBA_WWN PortWWN,
       HBA_UINT64 fcLUN,
       HBA_UINT8 EVPD,
       HBA_UINT32 PageCode,
       void * pRspBuffer,
       HBA_UINT32 RspBufferSize,
       void * pSenseBuffer,
       HBA_UINT32 SenseBufferSize
);
```

7.5.1.2 Description

The HBA_SendScsiInquiry function shall send a SCSI INQUIRY command (see SPC-3) to a remote FCP_Port (see FCP-2).

A SCSI command shall not be sent to an end port that does not have SCSI target functionality. A SCSI command shall not be sent if doing so would cause a SCSI overlapped command condition with a correctly operating target (see SAM-3). Proper use of tagged commands (see SAM-3) is an acceptable means of avoiding a SCSI overlapped command condition with targets that support tagged commands.

7.5.1.3 Arguments

handle shall be a handle to an open HBA.

PortWWN shall be the Port Name of a SCSI Target Port.

fcLUN shall be the SCSI LUN to send the SCSI INQUIRY command to.

EVPD shall be set to zero to return the standard SCSI INQUIRY data, or shall be set to one to return the vital product data specified by the page code.

PageCode shall be the Vital Product Data page code to request if EVPD is set to one, or shall be ignored if EVPD is set to zero.

pRspBuffer shall be a pointer to a buffer to receive the response.

RspBufferSize shall be the size in bytes of the buffer to receive response.

pSenseBuffer shall be a pointer to buffer to receive SCSI sense data.

SenseBufferSize shall be the size in bytes of the buffer to receive SCSI sense data.

7.5.1.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete payload of a reply to the SCSI INQUIRY command has been returned.
b) HBA_STATUS_ERROR_NOT_A_TARGET shall be returned to indicate the identified remote end port does not have SCSI Target functionality.
c) HBA_STATUS_ERROR_TARGET_BUSY shall be returned to indicate unable to send the requested command without causing a SCSI overlapped command condition.
d) HBA_STATUS_SCSI_CHECK_CONDITION shall be returned to indicate returned SCSI status indicates a SCSI CHECK CONDITION.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the response to the SCSI INQUIRY command. It shall be zero length if returned SCSI status indicates a SCSI CHECK CONDITION.

pSenseBuffer shall be unchanged. The buffer to which it points shall contain the SCSI sense data for the SCSI INQUIRY command. shall be nonzero length only if returned SCSI status indicates a SCSI CHECK CONDITION.

7.5.2 HBA_ScsiInquiryV2

7.5.2.1 Format

```
HBA_STATUS HBA_ScsiInquiryV2 (
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN discoveredPortWWN,
       HBA_UINT64 fcLUN,
       HBA_UINT8 CDB_Byte1,
       HBA_UINT8 CDB_Byte2,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize,
       HBA_UINT8 *pScsiStatus,
       void *pSenseBuffer,
       HBA_UINT32 *pSenseBufferSize
);
```

7.5.2.2 Description

The HBA_ScsiInquiryV2 function shall send a SCSI INQUIRY command to a remote end port (see SPC-3).

A SCSI command shall not be sent to an end port that does not have SCSI target functionality. A SCSI command shall not be sent if doing so would cause a SCSI overlapped command condition with a correctly operating target (see SAM-3). Proper use of tagged commands (see SAM-3) is an acceptable means of avoiding a SCSI overlapped command condition with targets that support tagged commands.

7.5.2.3 Arguments

handle shall be a handle to an open HBA through which the SCSI INQUIRY command shall be issued.

hbaPortWWN shall be the Port Name for a local HBA end port through which the SCSI INQUIRY command shall be issued.

discoveredPortWWN shall be the Port Name for an end port to which the SCSI INQUIRY command shall be sent.

fcLUN shall be the SCSI LUN to which the SCSI INQUIRY command shall be sent.

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 indicated in table 7

Table 7 — 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

CDB_Byte2 shall be the third byte of the CDB for the SCSI INQUIRY command. If CDB_Byte1 is 1, CDB_Byte2 shall be the Vital Product Data page code to request. If CDB_Byte1 is 2, 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.

pRspBuffer shall be a pointer to a buffer to receive the SCSI INQUIRY command response.

pRspBufferSize shall be a pointer to the size in bytes of the buffer to receive the SCSI INQUIRY command response.

pScsiStatus shall be a pointer to a buffer to receive SCSI status.

pSenseBuffer shall be a pointer to a buffer to receive SCSI sense data.

pSenseBufferSize shall be a pointer to the size in bytes of the buffer to receive SCSI sense data.

7.5.2.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete payload of a reply to the SCSI INQUIRY command has been returned.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR_NOT_A_TARGET shall be returned to indicate the identified remote end port does not have SCSI Target functionality.
d) HBA_STATUS_ERROR_TARGET_BUSY shall be returned to indicate unable to send the requested command without causing a SCSI overlapped command condition.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. If the function value is HBA_STATUS_OK, the buffer to which it points shall contain the response to the SCSI INQUIRY command.

pRspBufferSize shall be unchanged. The value of the integer to which it points 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.

pScsiStatus shall be unchanged. The value of the byte to which it points shall be the SCSI status (see SAM-3). 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 SCSI spec. A SCSI status of OK indicates a SCSI response is in the response buffer. A SCSI status of Check Condition indicates no value is stored in the response, and the sense buffer shall contain failure information if available. All other SCSI status codes should be interpreted by reference to SAM-3.

pSenseBuffer shall be unchanged. If the function value is HBA_STATUS_SCSI_CHECK_CONDITION, the buffer to which it points shall contain the sense data for the command.

pSenseBufferSize shall be unchanged. The value of the integer to which it points 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.

7.5.3 HBA_SendReportLUNs

7.5.3.1 Format

```
HBA_STATUS HBA_SendReportLUNs(
       HBA_HANDLE handle,
       HBA_WWN portWWN,
       void * pRspBuffer,
       HBA_UINT32 RspBufferSize,
       void * pSenseBuffer,
       HBA_UINT32 SenseBufferSize
);
```

7.5.3.2 Description

The HBA_SendReportLUNs function shall send a SCSI REPORT LUNS command (see SPC-3) to a remote FCP_Port (see FCP-2).

A SCSI command shall not be sent to an end port that does not have SCSI target functionality. A SCSI command shall not be sent if doing so would cause a SCSI overlapped command condition with a correctly operating target (see SAM-3). Proper use of tagged commands (see SAM-3) is an acceptable means of avoiding a SCSI overlapped command condition with targets that support tagged commands.

7.5.3.3 Arguments

handle shall be a handle to an open HBA.

PortWWN shall be the Port Name of a SCSI Target Port.

pRspBuffer shall be a pointer to a buffer to receive the response.

RspBufferSize shall be the size in bytes of the buffer to receive response.

pSenseBuffer shall be a pointer to buffer to receive SCSI sense data.

SenseBufferSize shall be the size in bytes of the buffer to receive SCSI sense data.

7.5.3.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete payload of a reply to the SCSI REPORT LUNS command has been returned.
b) HBA_STATUS_ERROR_NOT_A_TARGET shall be returned to indicate the identified remote end port does not have SCSI Target functionality.
c) HBA_STATUS_ERROR_TARGET_BUSY shall be returned to indicate unable to send the requested command without causing a SCSI overlapped command condition.
d) HBA_STATUS_SCSI_CHECK_CONDITION shall be returned to indicate returned SCSI status indicates a SCSI CHECK CONDITION.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the response to the SCSI REPORT LUNS command. It shall be zero length if returned SCSI status indicates a SCSI CHECK CONDITION.

pSenseBuffer shall be unchanged. The buffer to which it points shall contain the SCSI sense data for the SCSI REPORT LUNS command. shall be nonzero length only if returned SCSI status indicates a SCSI CHECK CONDITION.

7.5.4 HBA_ScsiReportLunsV2

7.5.4.1 Format

```
HBA_STATUS HBA_ScsiReportLUNsV2(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN discoveredPortWWN,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize,
       HBA_UINT8 *pScsiStatus,
       void *pSenseBuffer,
       HBA_UINT32 *pSenseBufferSize
);
```

7.5.4.2 Description

The HBA_SendReportLunsV2 function shall send a SCSI REPORT LUNS command to Logical Unit Number 0 of a remote end port (see SPC-3)

A SCSI command shall not be sent to an end port that does not have SCSI target functionality. A SCSI command shall not be sent if doing so would cause a SCSI overlapped command condition with a correctly operating target (see SAM-3). Proper use of tagged commands (see SAM-3) is an acceptable means of avoiding a SCSI overlapped command condition with targets that support tagged commands.

7.5.4.3 Arguments

handle shall be a handle to an open HBA through which the SCSI REPORT LUNS command shall be issued.

hbaPortWWN shall be the Port Name for a local HBA end port through which the SCSI REPORT LUNS command shall be issued.

discoveredPortWWN shall be the Port Name for an end port to which the SCSI REPORT LUNS command shall be sent.

pRspBuffer shall be a pointer to a buffer to receive the SCSI REPORT LUNS command response.

pRspBufferSize shall be a pointer to the size in bytes of the buffer to receive the SCSI REPORT LUNS command response.

pScsiStatus shall be a pointer to a buffer to receive SCSI status.

pSenseBuffer shall be a pointer to a buffer to receive SCSI sense data.

pSenseBufferSize shall be a pointer to the size in bytes of the buffer to receive SCSI sense data.

7.5.4.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete payload of a reply to the SCSI REPORT LUNS command has been returned.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR_NOT_A_TARGET shall be returned to indicate the identified remote end port does not have SCSI Target functionality.
d) HBA_STATUS_ERROR_TARGET_BUSY shall be returned to indicate unable to send the requested command without causing a SCSI overlapped command condition.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. If the function value is HBA_STATUS_OK, the buffer to which it points shall contain the response to the SCSI REPORT LUNS command.

pRspBufferSize shall be unchanged. The value of the integer to which it points 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.

pScsiStatus shall be unchanged. The value of the byte to which it points shall be the SCSI status (see SAM-3). 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 SCSI spec. A SCSI status of OK indicates a SCSI response is in the response buffer. A SCSI status of Check Condition indicates no value is stored in the response, and the sense buffer shall contain failure information if available. All other SCSI status codes should be interpreted by reference to SAM-3.

pSenseBuffer shall be unchanged. If the function value is HBA_STATUS_SCSI_CHECK_CONDITION, the buffer to which it points shall contain the sense data for the command.

pSenseBufferSize shall be unchanged. The value of the integer to which it points 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.

7.5.5 HBA_SendReadCapacity

7.5.5.1 Format

```
HBA_STATUS HBA_SendReadCapacity(
      HBA_HANDLE handle,
      HBA_WWN portWWN,
      HBA_UINT64 fcLUN,
      void * pRspBuffer,
      HBA_UINT32 RspBufferSize,
      void * pSenseBuffer,
      HBA_UINT32 SenseBufferSize
);
```

7.5.5.2 Description

The HBA_SendReadCapacity function shall send a SCSI READ CAPACITY command (see SBC-2) to a remote FCP_Port (see FCP-2).

A SCSI command shall not be sent to an end port that does not have SCSI target functionality. A SCSI command shall not be sent if doing so would cause a SCSI overlapped command condition with a correctly operating target (see SAM-3). Proper use of tagged commands (see SAM-3) is an acceptable means of avoiding a SCSI overlapped command condition with targets that support tagged commands.

7.5.5.3 Arguments

handle shall be a handle to an open HBA.

PortWWN shall be the Port Name of a SCSI Target Port.

fcLUN shall be the SCSI LUN to send the SCSI READ CAPACITY command to.

pRspBuffer shall be a pointer to a buffer to receive the response.

RspBufferSize shall be the size in bytes of the buffer to receive response data.

pSenseBuffer shall be a pointer to buffer to receive SCSI sense data.

SenseBufferSize shall be the size in bytes of the buffer to receive SCSI sense data.

7.5.5.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete payload of a reply to the SCSI READ CAPACITY command has been returned.
b) HBA_STATUS_ERROR_NOT_A_TARGET shall be returned to indicate the identified remote end port does not have SCSI Target functionality.
c) HBA_STATUS_ERROR_TARGET_BUSY shall be returned to indicate unable to send the requested command without causing a SCSI overlapped command condition.
d) HBA_STATUS_SCSI_CHECK_CONDITION shall be returned to indicate returned SCSI status indicates a SCSI CHECK CONDITION.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.

f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the response to the SCSI READ CAPACITY command. It shall be zero length if returned SCSI status indicates a SCSI CHECK CONDITION.

pSenseBuffer shall be unchanged. The buffer to which it points shall contain the SCSI sense data for the SCSI READ CAPACITY command. shall be nonzero length only if returned SCSI status indicates a SCSI CHECK CONDITION.

7.5.6 HBA_ScsiReadCapacityV2

7.5.6.1 Format

```
HBA_STATUS HBA_ScsiReadCapacityV2(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN discoveredPortWWN,
       HBA_UINT64 fcLUN,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize,
       HBA_UINT8 *pScsiStatus,
       void *pSenseBuffer,
       HBA_UINT32 *pSenseBufferSize
);
```

7.5.6.2 Description

The HBA_ScsiReadCapacityV2 function shall send a SCSI READ CAPACITY command to a remote end port (see SBC-2).

A SCSI command shall not be sent to an end port that does not have SCSI target functionality. A SCSI command shall not be sent if doing so would cause a SCSI overlapped command condition with a correctly operating target (see SAM-3). Proper use of tagged commands (see SAM-3) is an acceptable means of avoiding a SCSI overlapped command condition with targets that support tagged commands.

7.5.6.3 Arguments

handle shall be a handle to an open HBA through which the SCSI READ CAPACITY command shall be issued.

hbaPortWWN shall be the Port Name for a local HBA end port through which the SCSI READ CAPACITY command shall be issued.

discoveredPortWWN shall be the Port Name for an end port to which the SCSI READ CAPACITY command shall be sent.

fcLUN shall be the SCSI LUN to which the SCSI READ CAPACITY command shall be sent.

pRspBuffer shall be a pointer to a buffer to receive the SCSI READ CAPACITY command response.

pRspBufferSize shall be a pointer to the size in bytes of the buffer to receive the SCSI READ CAPACITY command response.

pScsiStatus shall be a pointer to a buffer to receive SCSI status.

pSenseBuffer shall be a pointer to a buffer to receive SCSI sense data.

pSenseBufferSize shall be a pointer to the size in bytes of the buffer to receive SCSI sense data.

7.5.6.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete payload of a reply to the SCSI READ CAPACITY command has been returned.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR_NOT_A_TARGET shall be returned to indicate the identified remote end port does not have SCSI Target functionality.
d) HBA_STATUS_ERROR_TARGET_BUSY shall be returned to indicate unable to send the requested command without causing a SCSI overlapped command condition.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. If the function value is HBA_STATUS_OK, the buffer to which it points shall contain the response to the SCSI READ CAPACITY command.

pRspBufferSize shall be unchanged. The value of the integer to which it points 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.

pScsiStatus shall be unchanged. The value of the byte to which it points shall be the SCSI status (see SAM-3). 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 SCSI spec. A SCSI status of OK indicates a SCSI response is in the response buffer. A SCSI status of Check Condition indicates no value is stored in the response, and the sense buffer shall contain failure information if available. All other SCSI status codes should be interpreted by reference to SAM-3.

pSenseBuffer shall be unchanged. If the function value is HBA_STATUS_SCSI_CHECK_CONDITION, the buffer to which it points shall contain the sense data for the command.

pSenseBufferSize shall be unchanged. The integer to which it points shall be set to 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.

7.6 SB Information Functions

7.6.1 HBA_GetSBTargetMapping

7.6.1.1 Format

```
HBA_STATUS HBA_GetSBTargetMapping (
       HBA_HANDLE          handle,
       HBA_WWN             hbaPortWWN,
       HBA_SBTARGETMAPPING *pMapping
);
```

7.6.1.2 Description

Retrieves the mappings offered by the specified local adapter port between OS identification of logical I/O devices and the worldwide-unique node-element identifier that is part of each device's Node Element Descriptor (NED). All SB-capable control units are required to support self-description data for the devices they manage, including device Node Element Descriptor data, as described in ANSI X3.296:1997, Single Byte Command Code Sets Connection (see SBCON).

7.6.1.3 Arguments

handle shall be an HBA_HANDLE to the open HBA containing the end port for which target mappings are requested

hbaPortWWN shall be the Name_Identifier the local adapter port for which target mappings are requested.

pMapping shall be a pointer to an HBA_SBTARGETMAPPING structure. The size of this structure is indicated by the NumberOfEntries value within the structure.

7.6.1.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned if the function was successful.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned if the adapter referenced by the handle does not contain an end port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned if the adapter referenced by the handle does not support target mapping.
d) HBA_STATUS_ERROR_MORE_DATA shall be returned if more space in the buffer is required to contain the mapping information.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pMapping shall be unchanged. The HBA_SBTARGETMAPPING structure to which it points shall now contain the full mapping information from OS identifications of I/O devices to SB identifications of devices for the specific local adapter port (see FC-SB-2, FC-SB-3 and relevant platform hardware and OS documentation). The value of the NumberOfEntries field of the returned structure shall indicate the total number of mappings the port has established even when the function returns an error because the buffer is too small to return all of them. An upper level 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 of sufficient size to accommodate the entire mapping structure.

7.6.2 HBA_GetSBStatistics

7.6.2.1 Format

```
HBA_STATUS HBA_GetSBStatistics (
       HBA_HANDLE          handle,
       const HBA_SBDEVID   *device,
       HBA_SBSTATISTICS    *statistics
);
```

7.6.2.2 Description

Returns statistics for a specific device provided via the SB protocol on a specific local adapter.

7.6.2.3 Arguments

handle shall be an HBA_HANDLE to an open HBA associated with the device for which SB statistics are requested

device shall be a pointer to an HBA_SBDEVID structure specifying the device for which statistics are requested.

statistics shall be a pointer to an HBA_SBSTATISTICS structure in which the statistics for the specified device may be returned.

7.6.2.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned if the function was successful.
b) HBA_STATUS_ERROR_INVALID_DEVICE shall be returned if the specified device is not accessible via the adapter referenced by the handle.
c) HBA_STATUS_ERROR_UNAVAILABLE shall be returned if statistics are not available for the specified device.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

statistics shall be unchanged. The structure to which it points shall contain the statistics for the specified device. The statistics returned may reflect activity on all paths via which the device is accessible.

7.7 SB Disk Device Information Functions

7.7.1 HBA_SBDskGetCapacity

7.7.1.1 Format

```
HBA_STATUS HBA_SBDskGetCapacity (
      HBA_DEVICESELFDESC  DeviceSelfDesc,
      HBA_SBDSKCAPACITY   *pSBDskCapacity
);
```

7.7.1.2 Description

Returns the capacity of the SB-attached disk device identified by DeviceSelfDesc. The information is returned in the specified HBA_SBDSKCAPACITY structure in two formats:

a) the same format as returned for SCSI disks by the SCSI Read Capacity command (number_of_blocks + block_size); and
b) the format typically used for disk capacities by IBM mainframe operating systems -- number_of_cylinders + number_of_tracks_per_cylinder + track_size.

The SCSI-format capacity information is derived from the cyls/tracks per cyl/track size format information. SBDskMaxUsableTrackLen is the unformatted capacity of the track. The capacity information returned by this function does not reflect the track capacity for user data, which depends on the size of the user data blocks on each track.

7.7.1.3 Arguments

DeviceSelfDesc shall specify the self description data (Token NED + Device NED) identifying the device for which capacity information is requested. ANSI X3.296:1997, Single Byte Command Code Sets Connection (see SBCON) describes Token NED and Device NED.

pSBDskCapacity shall be a pointer to an HBA_SBDSKCAPACITY structure in which capacity information for the specified device may be returned.

7.7.1.4 Return Values

f**unction value** shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned if the function was successful.
b) HBA_STATUS_ERROR_UNAVAILABLE shall be returned if the device identified by DeviceSelfDesc is unknown or no longer active.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pSBDskCapacity shall be unchanged. The structure to which it points shall contain the capacity information for the device.

7.8 Fabric Management Functions

7.8.1 HBA_SendCTPassThru

7.8.1.1 Format

```
HBA_STATUS HBA_SendCTPassThru(
       HBA_HANDLE handle,
       void *pReqBuffer,
       HBA_UINT32 ReqBufferSize,
       void *pRspBuffer,
       HBA_UINT32 RspBufferSize
);
```

7.8.1.2 Description

The HBA_SendCTPassThru function shall send a CT pass through frame. An HBA shall decode this CT_IU request per FC-GS-4, routing the CT frame in a Fabric according to the GS_TYPE field within the CT frame.

7.8.1.3 Arguments

handle shall be a handle to an open HBA.

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-4 with the byte ordering as defined in FC-FS.

ReqBufferSize shall be the size of the full CT payload including the CT header, in bytes.

pRspBuffer shall be a pointer to a buffer for the CT response.

RspBufferSize shall be the size of the buffer for the CT response payload in bytes.

7.8.1.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete reply to the CT Passthrough command has been returned.
b) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the CT response payload including the CT header received in response to the frame sent, as defined in FC-GS-4 with the byte ordering as defined in FC-FS. 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.

7.8.2 HBA_SendCTPassThruV2

7.8.2.1 Format

```
HBA_STATUS HBA_SendCTPassThruV2(
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       void *pReqBuffer,
       HBA_UINT32 ReqBufferSize,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize
);
```

7.8.2.2 Description

The HBA_SendCTPassThruV2 function shall send a CT request payload. An HBA should decode this CT_IU request per FC-GS-4, routing the CT frame in a fabric according to the GS_TYPE field within the CT frame.

7.8.2.3 Arguments

handle shall be a handle to an open HBA through which to issue the CT request.

hbaPortWWN shall be the Port Name of the local HBA Nx_Port through which to issue the CT request.

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-4 with the byte ordering as defined in FC-FS.

ReqBufferSize shall be the size of the full CT payload including the CT header, in bytes.

pRspBuffer shall be a pointer to a buffer for the CT response.

RspBufferSize shall be a pointer to the size of the buffer for the CT response payload in bytes.

7.8.2.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete reply to the CT Passthru command has been returned.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an Nx_Port with Port Name hbaPortWWN.
c) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the CT response payload including the CT header received in response to the frame sent, as defined in FC-GS-4 with the byte ordering as defined in FC-FS. 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.

pRspBufferSize shall be unchanged. The integer to which it points shall be set to the size (in bytes) of the actual response data.

7.8.3 HBA_SetRNIDMgmtInfo

7.8.3.1 Format

```
HBA_STATUS HBA_SetRNIDMgmtInfo(
       HBA_HANDLE handle,
       HBA_MGMTINFO info
);
```

7.8.3.2 Description

The HBA_SetRNIDMgmtInfo function shall set the RNID (Request Node Identification Information Data) returned from the HBA (see FC-FS).

7.8.3.3 Arguments

handle shall be a handle to an open HBA.

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-FS).

7.8.3.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the RNID reply information has been set as requested.
b) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

7.8.4 HBA_GetRNIDMgmtInfo

7.8.4.1 Format

```
HBA_STATUS HBA_GetRNIDMgmtInfo(
      HBA_HANDLE handle,
      HBA_MGMTINFO * pInfo
);
```

7.8.4.2 Description

The HBA_GetRNIDMgmtInfo function shall return the RNID (Request Node Identification Information Data) from the HBA (see FC-FS).

7.8.4.3 Arguments

handle shall be a handle to an open HBA.

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-FS).

7.8.4.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the RNID reply information has been returned.
b) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pInfo shall be unchanged. The structure to which it points shall contain the information that this HBA returns in a Specific Node Identification Data Format DFh of an RNID Accept (see FC-FS).

7.8.5 HBA_SendRNID

7.8.5.1 Format

```
HBA_STATUS HBA_SendRNID(
      HBA_HANDLE handle,
      HBA_WWN wwn,
      HBA_WWNTYPE wwntype,
      void * pRspBuffer,
      HBA_UINT32 * RspBufferSize
);
```

7.8.5.2 Description

Issues an RNID ELS to another end port with the Node Identification Data Format set to indicate the default Topology Discovery format (DFh) is to be returned (see FC-FS).

7.8.5.3 Arguments

handle shall be a handle to an open HBA.

wwn shall be the Port Name of the end port to which the RNID ELS shall be sent.

wwntype Deprecated.

pRspBuffer shall be a pointer to a buffer for the RNID response.

RspBufferSize shall be the size of the buffer for the RNID response payload in bytes.

7.8.5.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete reply to the RNID ELS has been returned, even if the response is an ELS reject response.
b) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the payload data from the RNID Reply as defined per FC-FS. 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.

7.8.6 HBA_SendRNIDV2

7.8.6.1 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
);
```

7.8.6.2 Description

The HBA_SendRNIDV2 function shall issue 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 x’00 00 00’, then the parameter shall be ignored. Otherwise, operation shall be as follows:

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 an entry in the discovered ports table, the RNID shall be sent.
b) If there is no entry in the discovered ports table for the destWWN or destFCID, then the RNID shall be sent.
c) If there is an entry in the discovered ports table for the destWWN, but the destFCID does not match, then the request shall be rejected.
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 WWN 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. If the Common Identification Data Length is zero in the RNID response, then this test shall be omitted.

7.8.6.3 Arguments

handle shall be a handle to an open HBA through which the ELS shall be sent.

hbaPortWWN shall be the Port Name of the local HBA end port through which the ELS shall be sent.

destWWN shall be the Port Name of the remote FC_Port to which the ELS shall be sent.

destFCID shall be the address identifier of the destination to which the ELS is sent if destFCID is nonzero. destFCID shall be ignored if destFCID is zero.

NodeIdDataFormat shall be a valid value for Node Identification Data Format as per FC-FS.

pRspBuffer shall be a pointer to a buffer to receive the ELS response.

pRspBufferSize shall be a pointer to the size in bytes of pRspBuffer.

7.8.6.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete LS_ACC to the RNID ELS has been returned.
b) HBA_STATUS_ERROR_ELS_REJECT shall be returned to indicate the RNID ELS was rejected by the destination end port.
c) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
d) HBA_STATUS_ERROR_ILLEGAL_FCID shall be returned to indicate the destWWN/destFCID pair conflicts with a discovered Port Name/address identifier pair known by the HBA referenced by handle.
e) HBA_STATUS_ERROR_ILLEGAL_FCID shall be returned to indicate the N_Port WWN in the RNID response does not match the destWWN.
f) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
g) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the payload data from the RNID Reply as defined per FC-FS. Note, if the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size specified in argument pRspBufferSize at entry to the function, the returned data shall be truncated to the size specified in the argument.

pRspBufferSize shall be unchanged. The integer to which it points shall contain the size in bytes of the complete ELS reply payload. This may exceed the size specified as an argument. This shall indicate that the data in pRspBuffer has been truncated.

7.8.7 HBA_SendRPL

7.8.7.1 Format

```
HBA_STATUS HBA_SendRPL (
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN agent_wwn,
       HBA_UINT32 agent_domain,
       HBA_UINT32 portIndex,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize
);
```

7.8.7.2 Description

The HBA_SendRPL function shall issue a Read Port List (RPL) Extended Link Service through the specified HBA to a specified end port or domain controller (see FC-FS).

7.8.7.3 Arguments

handle shall be a handle to an open HBA through which the ELS shall be sent.

hbaPortWWN shall be the Port Name of the local HBA end port through which the ELS shall be sent.

agent_wwn shall be the Port Name of an FC_Port that shall be requested to provide its list of FC_Ports if agent_wwn is nonzero. agent_wwn shall be ignored if agent_wwn is zero.

agent_domain shall be a domain number and the domain controller for that domain shall be the entity that shall be requested to provide its list of FC_Ports if agent_wwn is zero. agent_domain shall be ignored if agent_wwn is nonzero.

portIndex shall be the index of the first FC_Port requested in the response list.

NOTE 14 If the recipient complies with FC-FS, the index of the first FC_Port in the complete list maintained by the recipient of the request is zero.

pRspBuffer shall be a pointer to a buffer to receive the ELS response.

pRspBufferSize shall be a pointer to the size in bytes of pRspBuffer.

NOTE 15 If the responding entity complies with FC-FS, it truncates the list in the response to the number of FC_Ports that fit.

7.8.7.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete LS_ACC to the RPL ELS has been returned.
b) HBA_STATUS_ERROR_ELS_REJECT shall be returned to indicate the RPL ELS was rejected by the destination end port.
c) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.

e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the payload data from the RPL Reply as defined per FC-FS. If the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size specified in argument pRspBufferSize at entry to the function, the returned data shall be truncated to the size specified in the argument.

pRspBufferSize shall be unchanged. The integer to which it points shall contain the size in bytes of the complete ELS reply payload. This may exceed the size specified as an argument. This shall indicate that the data in pRspBuffer has been truncated.

NOTE 16 Truncation is not necessary if the responding entity complies with FC-FS.

7.8.8 HBA_SendRPS

7.8.8.1 Format

```
HBA_STATUS HBA_SendRPS (
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN agent_wwn,
       HBA_UINT32 agent_domain,
       HBA_WWN object_wwn,
       HBA_UINT32 object_port_number,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize
);
```

7.8.8.2 Description

The HBA_SendRPS function shall issue a Read Port Status Block (RPS) Extended Link Service through the specified HBA to a specified FC_Port or domain controller (see FC-FS).

7.8.8.3 Arguments

handle shall be a handle to an open HBA through which the ELS shall be sent.

hbaPortWWN shall be the Port Name of the local HBA end port through which the ELS shall be sent.

agent_wwn shall be the Port Name of an FC_Port that shall be requested to provide Port Status if agent_wwn is nonzero. agent_wwn shall be ignored if agent_wwn is zero.

agent_domain shall be the domain number for the domain controller that shall be requested to provide Port status if agent_wwn is zero. agent_domain shall be ignored if agent_wwn is nonzero.

object_wwn shall be the Port Name of an FC_Port for which Port Status shall be returned if object_wwn is nonzero. object_wwn shall be ignored if object_wwn is zero

object_port_number shall be a relative port number of the FC_Port for which Port Status shall be returned if object_wwn is zero. Relative port number shall be defined in a vendor specific manner within the entity to which the request is sent. object_port_number shall be ignored if object_wwn is nonzero.

pRspBuffer shall be a pointer to a buffer to receive the ELS response.

pRspBufferSize shall be a pointer to the size in bytes of pRspBuffer. A size of 56 is sufficient for the largest response.

7.8.8.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete LS_ACC to the RPS ELS has been returned.
b) HBA_STATUS_ERROR_ELS_REJECT shall be returned to indicate the RPS ELS was rejected by the destination end port.
c) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the payload data from the RPS Reply as defined per FC-FS. If the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size specified in argument pRspBufferSize at entry to the function, the returned data shall be truncated to the size specified in the argument.

pRspBufferSize shall be unchanged. The integer to which it points shall contain the size in bytes of the complete ELS reply payload. This may exceed the size specified as an argument. This shall indicate that the data in pRspBuffer has been truncated.

7.8.9 HBA_SendSRL

7.8.9.1 Format

```
HBA_STATUS HBA_SendSRL (
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN wwn,
       HBA_UINT32 domain,
       void *pRspBuffer,
       HBA_UINT32 *pRspBufferSize
);
```

7.8.9.2 Description

The HBA_SendSRL function shall issue a Scan Remote Loop (SRL) Extended Link Service through the specified HBA to a specified domain controller (see FC-FS).

7.8.9.3 Arguments

handle shall be a handle to an open HBA through which the ELS shall be sent.

hbaPortWWN shall be the Port Name of the local HBA end port through which the ELS shall be sent.

wwn shall be the Port Name 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. wwn shall be ignored if wwn is zero.

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. domain shall be ignored if wwn is nonzero.

pRspBuffer shall be a pointer to a buffer to receive the ELS response.

pRspBufferSize shall be a pointer to the size in bytes of pRspBuffer. Eight is a sufficient length for any response.

7.8.9.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete LS_ACC to the SRL ELS has been returned.
b) HBA_STATUS_ERROR_ELS_REJECT shall be returned to indicate the SRL ELS was rejected by the destination domain.
c) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the payload data from the SRL Reply as defined per FC-FS. Note, if the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size specified in argument pRspBufferSize at entry to the function, the returned data shall be truncated to the size specified in the argument.

pRspBufferSize shall be unchanged. The integer to which it points shall contain the size in bytes of the complete ELS reply payload. This may exceed the size specified as an argument. This shall indicate that the data in pRspBuffer has been truncated.

7.8.10 HBA_SendLIRR

7.8.10.1 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
);
```

7.8.10.2 Description

The HBA_SendLIRR function shall issue a Link Incident Record Registration (LIRR) Extended Link Service through the specified HBA end port to a specified remote end port (see FC-FS). The HBA and its software shall not autonomously originate LIRR, so link incident registration shall be entirely under control of application software.

7.8.10.3 Arguments

handle shall be a handle to an open HBA through which the ELS shall be sent.

hbaPortWWN shall be the Port Name of the local HBA end port through which the ELS shall be sent.

destWWN shall be the Port Name of the remote FC_Port to which the ELS shall be sent. If this is zero, the destination shall be the management server well known address.

function shall be the code for the registration function to be performed. See FC-FS for permitted values and their meanings.

type shall be the FC-4 device TYPE for which specific link incident information requested if type is nonzero. Only the common link incident information is requested if type is zero.

pRspBuffer shall be a pointer to a buffer to receive the ELS response.

pRspBufferSize shall be a pointer to the size in bytes of pRspBuffer. Eight is a sufficient length for any response.

7.8.10.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete LS_ACC to the LIRR ELS has been returned.
b) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the local HBA does not support link incident reporting.
c) HBA_STATUS_ERROR_ELS_REJECT shall be returned to indicate the LIRR ELS was rejected by the destination end port.
d) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
e) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
f) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the payload data from the LIRR Reply as defined per FC-FS. Note, if the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size specified in argument pRspBufferSize at entry to the function, the returned data shall be truncated to the size specified in the argument.

pRspBufferSize shall be unchanged. The integer to which it points shall contain the size in bytes of the complete ELS reply payload. This may exceed the size specified as an argument. This shall indicate that the data in pRspBuffer has been truncated.

7.8.11 HBA_SendRLS

7.8.11.1 Format

```
HBA_STATUS HBA_SendRLS (
      HBA_HANDLE handle,
      HBA_WWN hbaPortWWN,
      HBA_WWN destWWN,
      void *pRspBuffer,
      HBA_UINT32 *pRspBufferSize
);
```

7.8.11.2 Description

The HBA_SendRLS function shall issue 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 Port Name (see FC-FS).

7.8.11.3 Arguments

handle shall be a handle to an open HBA through which the ELS shall be sent.

hbaPortWWN shall be the Port Name of the local HBA end port through which the ELS shall be sent.

destWWN shall be the Port Name of the remote FC_Port to which the ELS shall be sent.

pRspBuffer shall be a pointer to a buffer to receive the ELS response.

pRspBufferSize shall be a pointer to the size in bytes of pRspBuffer. A size of 28 is sufficient for the largest response.

7.8.11.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate the complete LS_ACC to the RLS ELS has been returned.
b) HBA_STATUS_ERROR_ELS_REJECT shall be returned to indicate the RNID ELS was rejected by the destination FC_Port.
c) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name hbaPortWWN.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pRspBuffer shall be unchanged. The buffer to which it points shall contain the payload data from the RLS Reply as defined per FC-FS. Note, if the ELS was rejected, this shall be the LS_RJT payload. If the size of the reply payload exceeds the size specified in argument pRspBufferSize at entry to the function, the returned data shall be truncated to the size specified in the argument.

pRspBufferSize shall be unchanged. The integer to which it points shall contain the size in bytes of the complete ELS reply payload. This may exceed the size specified as an argument. This shall indicate that the data in pRspBuffer has been truncated.

7.9 Event Handling Functions

7.9.1 Polled Event Reporting Behavior Model

The polled event reporting method provides a simple polled interface to a basic set of HBA-detected events. It is retained in this standard for compatibility with the Common HBA API Interface specification in FC-MI. It comprises a single function, HBA_GetEventBuffer.

An implementation of the polled event reporting method shall behave as though it were a circular queue of event records in the format of HBA_EventInfo structures (see 6.9.1.2) that shall represent RSCN, link status, or other events. Event records shall be added to the presumed circular queue as events occur. Events shall be removed from the presumed circular queue in order of occurrence as any application calls HBA_GetEventBuffer. The size of the queue shall be implementation dependent. If the queue becomes full, any newly added records shall replace the oldest records and the next record to be delivered to an application shall be the oldest remaining record. This shall cause the oldest records to be lost.

If multiple applications make overlapping sequences of HBA_GetEventBuffer calls, each available event record shall each be delivered in the return to only one HBA_GetEventGuffer call. In this circumstance, the exact distribution of records to applications may not be predictable, but the sequence of events delivered to any application shall be strictly in order of event occurrence.

Any event reported via the polled event reporting method shall also be reported to all applications that have registered for that event through the asynchronous event reporting method.

The arrival of an RSCN ELS shall be treated as a separate event for each Affected Port_ID Page carried by the RSCN.

7.9.2 HBA_GetEventBuffer

7.9.2.1 Format

```
HBA_STATUS HBA_GetEventBuffer(
       HBA_HANDLE handle,
       HBA_EVENTINFO *pEventBuffer,
       HBA_UINT32 *pEventCount
);
```

7.9.2.2 Description

The HBA_GetEventBuffer function shall remove and return the next events from the HBA's event queue. The number of events returned shall be the lesser of the value of argument EventCount at call and the number of entries available in the event queue.

7.9.2.3 Arguments

handle shall be a handle to an open HBA.

pEventBuffer shall be a pointer to a buffer to receive events.

pEventCount shall be a pointer to the number of event records that fit in the space allocated for the buffer to receive events. It shall be set to the size (in event records) of the buffer for receiving events on call, and shall be returned as the number of events actually delivered.

7.9.2.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate no errors were encountered and pEventCount indicates the number of event records returned.
b) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pEventBuffer shall be unchanged. The buffer to which it points shall contain event records representing previously undelivered events.

pEventCount shall be unchanged. The integer to which it points shall contain the number of event records that actually were delivered.

7.9.3 Overview of Asynchronous Event Reporting

7.9.3.1 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 promptly after detection of the event; however, this standard places no specific limits on promptness.

For the purpose of asynchronous event reporting, events detected by an HBA shall be grouped coarsely into event categories and more specifically into event types. When an application registers a callback function for asynchronous event notification, it selects a category and source of events that shall be reported via that callback function. Depending on the registration function, the source may be the local system, a local HBA, a local HBA port, or an FCP target device. An application that is registered for events of a selected category and source shall be notified of every event detected concerning the selected source of any type in the selected category. An application shall not be notified of any event of a category and source for which it is not registered at the time of occurrence of the event.

An application that has registered for notification of events of a selected category and source may later deregister for notification of that category and source of events.

If an application registers a callback function for an event category and source without explicitly deregistering previous callback functions for the same event category and source, each registered function shall be called on occurrence of any event of the selected category and source.

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. Any event reported via the polled event reporting method shall also be reported via the asynchronous event reporting method to all applications that have registered for that event.

The arrival of an RSCN ELS shall be treated as a separate event for each Affected Port_ID Page carried by the RSCN.

7.9.3.2 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. The possibility arises that 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:

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.

If all instances of HBA specific software indicate the same result, the HBA API library shall return a status appropriate to that result.

If any instance of HBA specific software indicated successful registration, the HBA API library shall return HBA_STATUS_OK.

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.

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.

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 17 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.

7.9.4 HBA_RegisterForAdapterAddEvents

7.9.4.1 Format

```
HBA_STATUS HBA_RegisterForAdapterAddEvents(
       void (*pCallback) (
              void *pData,
              HBA_WWN PortWWN,
              HBA_UINT32 eventType
       ),
       void *pUserData,
       HBA_CALLBACKHANDLE *pCallbackHandle
       );
```

7.9.4.2 Description

The HBA_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 a Port Name of the new HBA. The event type shall be HBA_EVENT_ADAPTER_ADD. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.4.3 Arguments

pCallback shall be a pointer to the entry to the callback routine.

pUserData shall be a pointer that shall be passed to the callback routine with each event. This may be used for correlating the event with the source of its event registration.

pCallbackHandle shall be a pointer to a structure in which an opaque identifier that may be used to deregister the callback may be returned.

7.9.4.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate successful callback function registration.
b) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pCallbackHandle shall be unchanged. The structure to which it points shall contain an opaque identifier that may be used to deregister the callback if the function value is HBA_STATUS_OK.

7.9.4.5 Callback Arguments

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.

PortWWN shall be the Port Name of any end port on the HBA that was added.

eventType shall be HBA_EVENT_ADAPTER_ADD

7.9.5 HBA_RegisterForAdapterEvents

7.9.5.1 Format

```
HBA_STATUS HBA_RegisterForAdapterEvents(
       void (*pCallback) (
              void *pData,
              HBA_WWN PortWWN,
              HBA_UINT32 eventType
       ),
       void *pUserData,
       HBA_HANDLE handle,
       HBA_CALLBACKHANDLE *pCallbackHandle
);
```

7.9.5.2 Description

The HBA_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 adapter, the callback function shall be called with event type of HBA_EVENT_ADAPTER_REMOVE or HBA_EVENT_ADAPTER_CHANGE. Events shall cause callbacks whether the HBA handle specified at registration is held open or not. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.5.3 Arguments

pCallback shall be a pointer to the entry to the callback routine.

pUserData shall be a pointer that shall be passed to the callback routine with each event. This may be used for correlating the event with the source of its event registration.

handle shall be a handle to an open HBA for which event callbacks are requested.

pCallbackHandle shall be a pointer to a structure in which an opaque identifier that may be used to deregister the callback may be returned.

7.9.5.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate successful callback function registration.
b) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pCallbackHandle shall be unchanged. The structure to which it points shall contain an opaque identifier that may be used to deregister the callback if the function value is HBA_STATUS_OK.

7.9.5.5 Callback Arguments

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.

PortWWN shall be a Port Name of any end port on the HBA that detected the event. The client should re-discover all aspects of the HBA and ALL connected FC_Ports as the prior state may not be accurate.

eventType shall be a value specified in 6.10.1.3 indicating the type of event that occurred.

7.9.6 HBA_RegisterForAdapterPortEvents

7.9.6.1 Format

```
HBA_STATUS HBA_RegisterForAdapterPortEvents(
	void (*pCallback) (
		void *pData,
		HBA_WWN PortWWN,
		HBA_UINT32 eventType,
		HBA_UINT32 fabricPortID
	),
	void *pUserData,
	HBA_HANDLE handle,
	HBA_WWN PortWWN,
	HBA_CALLBACKHANDLE *pCallbackHandle
);
```

7.9.6.2 Description

The HBA_RegisterForAdapterPortEvents function shall register an application defined function that shall be called upon occurrence of 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. Event types shall be HBA_EVENT_PORT_OFFLINE, HBA_EVENT_PORT_ONLINE, HBA_EVENT_PORT_NEW_TARGETS, HBA_EVENT_PORT_FABRIC or HBA_EVENT_PORT_UNKNOWN. 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, as per the RSCN definition in FC-FS. The arrival of an RSCN ELS shall be treated as a separate event for each Affected Port ID Page carried by the RSCN. For all other event types, fabricPortID shall be ignored. Events shall cause callbacks whether the HBA handle specified at registration is held open or not. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.6.3 Arguments

pCallback shall be a pointer to the entry to the callback routine.

pUserData shall be a pointer that shall be passed to the callback routine with each event. This may be used for correlating the event with the source of its event registration.

handle shall be a handle to an open HBA for which event callbacks are requested.

PortWWN shall be the Port Name of the end port on the specified HBA for which event callbacks are requested

pCallbackHandle shall be a pointer to a structure in which an opaque identifier that may be used to deregister the callback may be returned.

7.9.6.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate successful callback function registration.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name PortWWN.
c) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pCallbackHandle shall be unchanged. The structure to which it points shall contain an opaque value that may be used to deregister the callback if the function value is HBA_STATUS_OK.

7.9.6.5 Callback Arguments

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.

PortWWN shall be the Port Name of the HBA end port that detected the event.

eventType shall be a value specified in 6.10.1.4 indicating the type of event that occurred.

fabricPortID If the event is of type HBA_EVENT_PORT_FABRIC, this shall contain the RSCN affected Port ID page, as per the RSCN definition in FC-FS. For all other event types, fabricPortID shall be ignored.

7.9.7 HBA_RegisterForAdapterPortStatEvents

7.9.7.1 Format

```
HBA_STATUS HBA_RegisterForAdapterPortStatEvents(
       void (*pCallback) (
              void *pData,
              HBA_WWN PortWWN,
              HBA_UINT32 eventType,
       ),
       void *pUserData,
       HBA_HANDLE handle,
       HBA_WWN PortWWN,
       HBA_PortStatistics stats,
       HBA_UINT32 statType,
       HBA_CALLBACKHANDLE *pCallbackHandle
);
```

7.9.7.2 Description

The HBA_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 HBA statistics category asynchronous event so 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. Events shall cause callbacks whether the HBA handle specified at registration is held open or not. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.7.3 Arguments

pCallback shall be a pointer to the entry to the callback routine.

pUserData shall be a pointer that shall be passed to the callback routine with each event. This may be used for correlating the event with the source of its event registration.

handle shall be a handle to an open HBA for which event callbacks are requested.

PortWWN shall be the Port Name of the end port on the specified HBA for which event callbacks are requested

stats shall be a Port Statistics 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 watch for. If statType is HBA_EVENT_PORT_STAT_GROWTH, any non-null values in the stats structure shall be interpreted as growth rate numbers over 1 minute, although the frequency at which the growth is monitored is vendor specific

statType shall be a value specified in 6.10.1.5 that shall determine whether the events registered by this call are threshold crossing or growth rate of the indicated counters

pCallbackHandle shall be a pointer to a structure in which an opaque identifier that may be used to deregister the callback may be returned.

7.9.7.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate successful callback function registration.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name PortWWN.
c) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA specific library or underlying system does not support statistic events.
d) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
e) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pCallbackHandle shall be unchanged. The structure to which it points shall contain an opaque identifier that may be used to deregister the callback if the function value is HBA_STATUS_OK.

7.9.7.5 Callback Arguments

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.

PortWWN shall be the Port Name of the HBA end port that detected the event.

eventType shall be a value specified in 6.10.1.5 indicating the type of event that occurred.

7.9.8 HBA_RegisterForTargetEvents

7.9.8.1 Format

```
HBA_STATUS HBA_RegisterForTargetEvents(
       void (*pCallback) (
              void *pData,
              HBA_WWN hbaPortWWN,
              HBA_WWN discoveredPortWWN,
              HBA_UINT32 eventType,
       ),
       void *pUserData,
       HBA_HANDLE handle,
       HBA_WWN hbaPortWWN,
       HBA_WWN discoveredPortWWN,
       HBA_CALLBACKHANDLE *pCallbackHandle,
       HBA_UINT32 allTargets
);
```

7.9.8.2 Description

The HBA_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-2 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. Events shall cause callbacks whether the HBA handle specified at registration is held open or not. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.8.3 Arguments

pCallback shall be a pointer to the entry to the callback routine.

pUserData shall be a pointer that shall be passed to the callback routine with each event. This may be used for correlating the event with the source of its event registration.

handle shall be a handle to an open HBA for which event callbacks are requested.

hbaPortWWN shall be the Port Name of the end port on the specified HBA for which event callbacks are requested

discoveredPortWWN shall be the Port Name of the target end port for which event callbacks are requested

pCallbackHandle shall be a pointer to a structure in which an opaque identifier that may be used to deregister the callback may be returned.

allTargets shall indicate the scope of target end ports registered by this call. If allTargets is non-zero, the value in discoveredPortWWN shall be ignored, and events for all current and future discovered target end ports shall be registered by this call. If allTargets is zero, only event for the target end port specified by discoveredPortWWN shall be registered by this call

7.9.8.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate successful callback function registration.
b) HBA_STATUS_ERROR_ILLEGAL_WWN shall be returned to indicate the HBA referenced by handle does not contain an end port with Port Name PortWWN.
c) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pCallbackHandle shall be unchanged. The structure to which it points shall contain an opaque identifier that may be used to deregister the callback if the function value is HBA_STATUS_OK.

7.9.8.5 Callback Arguments

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.

hbaPortWWN shall be the Port Name of the local HBA end port through which the target event was detected

discoveredPortWWN shall be the Port Name of the remote end port at which the target event was detected

eventType shall be a value specified in 6.10.1.6 indicating the type of event that occurred.

7.9.9 HBA_RegisterForLinkEvents

7.9.9.1 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,
);
```

7.9.9.2 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. Events shall cause callbacks whether the HBA handle specified at registration is held open or not. To terminate event delivery, HBA_RemoveCallback shall be called.

7.9.9.3 Arguments

pCallback shall be a pointer to the entry to the callback routine.

pUserData shall be a pointer that shall be passed to the callback routine with each event. This may be used for correlating the event with the source of its event registration.

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.

RLIRBufferSize shall be the size in bytes of the buffer that pRLIRBuffer addresses.

handle shall be a handle to an open HBA for which event callbacks are requested.

pCallbackHandle shall be a pointer to a structure in which an opaque identifier that may be used to deregister the callback may be returned.

7.9.9.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate successful callback function registration.
b) HBA_STATUS_ERROR_NOT_SUPPORTED shall be returned to indicate the HBA specific library or underlying system does not support statistic events.
c) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.

d) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

pCallbackHandle shall be unchanged. The structure to which it points shall contain an opaque identifier that may be used to deregister the callback if the function value is HBA_STATUS_OK.

7.9.9.5 Callback Arguments

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.

adapterWWN shall be the Port Name of any end port on the HBA from which a fabric link event is being reported

eventType shall be a value specified in 6.10.1.7 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.

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, as defined per FC-FS. 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.

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.

7.9.10 HBA_RemoveCallback

7.9.10.1 Format

```
HBA_STATUS HBA_RemoveCallback(
      HBA_CALLBACKHANDLE callbackHandle
);
```

7.9.10.2 Description

The HBA_RemoveCallback function shall remove an instance of a callback routine specified by the opaque handle callbackHandle.

7.9.10.3 Arguments

callbackHandle shall be the opaque handle returned by the asynchronous event registration function that shall be deregistered.

7.9.10.4 Return Values

function value shall be a value defined in 6.2 indicating the reason for completion of the requested function:

a) HBA_STATUS_OK shall be returned to indicate successful callback function removal.
b) HBA_STATUS_ERROR may be returned to indicate any problem with no required value.
c) The value among those in 6.2 for which the comment most closely describes the result of the function should be returned to indicate any reason with no required value.

8 Configuration

8.1 Overview

This clause applies only to HBA API libraries with OS independent structure. No part of it applies to HBA API libraries with OS specific structure. It specifies a uniform, complete, and persistent inventory of the components that compose an HBA API on a system and their relationships to one another. It is intended to facilitate both configuration services and HBA API implementations. It refers to features of specific operating systems.

A given environment using the HBA API may be the result of several installation processes from various vendors. Each vendor's installation process may install one or more HBA specific libraries and may install a version of the Common API library (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 shall be of the format

\HKEY_LOCAL_MACHINE\SOFTWARE\SNIA\HBA\vendorid

b) The key shall have a value named LibraryFile of type REG_SZ that contains the full path to the vendor's library.
c) *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.

Example:

```
\\HKEY_LOCAL_MACHINE\SOFTWARE\SNIA\HBA\org.snia.sample
LibraryFile = “c:/Program Files/Samplevendor/Library.dll”
```

The method used to load multiple vendors' libraries in a Win32 environment shall include these procedures:

a) The wrapper library shall be installed as files named HBAAPI.DLL and HBAAPI.LIB and shall be installed in directory %systemroot%/System32/.
b) The wrapper library shall read the registry to discover HBA specific library names.
c) Using the Win32 routines LoadLibrary and GetProcAddress, the wrapper shall open and discover the appropriate vendors libraries.
d) The wrapper library shall use these libraries to discover the aggregate number of adapters and report this to the upper level application.
e) The names of the lower level adapters shall be passed through the wrapper library.
f) 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.
g) 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 shall be:

a) If it does not already exist, a text file /etc/hba.conf shall be created.
b) In the file /etc/hba.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

c) *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.
d) <*sp*> shall be any nonnull combination of space characters and tab characters.
e) *vslpath* shall be the full path to the HBA specific library.

Examples:

```
org.snia.sample    /usr/lib/libhbaapi-reference-vsl.so
com.hotbiscuitsadapters.supervsl    /usr/lib/sparcv9/lib-hba-supervsl.so
```

The method used to load multiple vendors' libraries in a unix environment shall include the procedures in the following list:

a) The wrapper library shall be installed as files named HBAAPI.DLL and HBAAPI.LIB and shall be installed in the directory appropriate to the library type:
   A) 32-bit: /usr/lib
   B) 64-bit: vendor-specific subdirectory of /usr/lib/ for 64-bit libraries (e.g., /usr/lib/sparcv9/).
b) The wrapper library shall read the /etc/hba.conf file to discover HBA specific library names.
c) Using OS HBA specific library loading routines (e.g., dlopen and dlsym on Solaris), the wrapper shall open and discover the appropriate vendors libraries.
d) The wrapper library shall use these libraries to discover the aggregate number of adapters and report this to the upper level application.
e) The names of the lower level adapters shall be passed through the wrapper library.
f) 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.
g) Remaining calls shall be routed by the wrapper library to the appropriate HBA specific library given the HBA_HANDLE.

Annex A
(Normative)

FC-HBA Compliance Requirements

A.1 Overview

FC-HBA compliant software 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 FC-HBA compliant software shall implement and how it shall indicate optional features that it does not implement. This normative annex also identifies additional features that software that is compliant with FC-HBA Management Interoperability Extensions shall implement.

Functions shall be mandatory, management-mandatory, optional, or not allowed.

A management-mandatory function shall be optional for compliance with FC-HBA but mandatory for compliance with the FC-HBA Management Interoperability Extensions.

A compliant HBA specific library shall:

a) for each mandatory function, 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, 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 18 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 HBAperform the functions and return the response as specified in this standard.

Attributes and statistics shall be mandatory, management-mandatory, optional, or not allowed. A management-mandatory attribute or statistic shall be optional for compliance with FC-HBA but mandatory for compliance with the FC-HBA Management Interoperability Extensions. 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 and table A.2 specify the the requirements for implementing functions for software that is compliant with FC-HBA and software that is compliant with FC-HBA Management Interoperability Extensions. Table A.1 specifies requirements for all implementations of compliant software. The functions in table A.2 shall be not allowed for all implementations of compliant software on systems that do not support SB devices, and shall be required as specified in the table for all implementations of compliant software on systems that support SB devices. Within each table, different requirements are specified for an HBA API library than for HBA specific libraries.

Table A.1 — General Function Requirements

Function	**For HBA API library**	**For HBA specific libraries**	**Reference**
Library Control Functions			
HBA_GetVersion	M	M	7.2.1
HBA_LoadLibrary	M	M	7.2.2
HBA_FreeLibrary	M	M	7.2.3
HBA_RegisterLibrary	N	M	7.2.4
HBA_RegisterLibraryV2	N	M	7.2.5
HBA_GetWrapperLibraryAttributes	M	N	7.2.6
HBA_GetVendorLibraryAttributes	M	M	7.2.7
HBA_GetNumberOfAdapters	M	M	7.2.8
HBA_RefreshInformation	M	M	7.2.9
HBA_RefreshAdapterConfiguration	M	N	7.2.10
HBA_ResetStatistics	M	O	7.2.11
Adapter and Port Information Functions			
HBA_GetAdapterName	M	M	7.3.1
HBA_OpenAdapter	M	M	7.3.2
HBA_OpenAdapterByWWN	M	O	7.3.3
HBA_CloseAdapter	M	M	7.3.4
HBA_GetAdapterAttributes	M	M	7.3.5
HBA_GetAdapterPortAttributes	M	M	7.3.6
HBA_GetDiscoveredPortAttributes	M	M	7.3.7
HBA_GetPortAttributesByWWN	M	O	7.3.8
HBA_GetPortStatistics	M	M	7.3.9
HBA_GetFC4Statistics	M	O	7.3.10

Key:
M designates a function that is mandatory.
Mgt designates a function that is management-mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Table A.1 — General Function Requirements

Function	For HBA API library	For HBA specific libraries	Reference
FCP Information Functions			
HBA_GetBindingCapability	M	O	7.4.1
HBA_GetBindingSupport	M	O	7.4.2
HBA_SetBindingSupport	M	O	7.4.3
HBA_GetFcpTargetMapping	M	O	7.4.4
HBA_GetFcpTargetMappingV2	M	O	7.4.5
HBA_GetFcpPersistentBinding	M	O	7.4.6
HBA_GetPersistentBindingV2	M	O	7.4.7
HBA_SetPersistentBindingV2	M	O	7.4.8
HBA_RemovePersistentBinding	M	O	7.4.9
HBA_RemoveAllPersistentBindings	M	O	7.4.10
HBA_GetFCPStatistics	M	O	7.4.11
SCSI Information Functions			
HBA_SendScsiInquiry	M	O	7.5.1
HBA_ScsiInquiryV2	M	O	7.5.2
HBA_SendReportLUNs	M	O	7.5.3
HBA_ScsiReportLunsV2	M	O	7.5.4
HBA_SendReadCapacity	M	O	7.5.5
HBA_ScsiReadCapacityV2	M	O	7.5.6
Fabric Management Functions			
HBA_SendCTPassThru	M	Mgt	7.8.1
HBA_SendCTPassThruV2	M	Mgt	7.8.2
HBA_SetRNIDMgmtInfo	M	O	7.8.3
HBA_GetRNIDMgmtInfo	M	M	7.8.4
HBA_SendRNID	M	Mgt	7.8.5
HBA_SendRNIDV2	M	Mgt	7.8.6
HBA_SendRPL	M	Mgt	7.8.7
HBA_SendRPS	M	Mgt	7.8.8
HBA_SendSRL	M	Mgt	7.8.9
HBA_SendLIRR	M	Mgt	7.8.10

Key:
M designates a function that is mandatory.
Mgt designates a function that is management-mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Table A.1 — General Function Requirements

Function	For HBA API library	For HBA specific libraries	Reference
HBA_SendRLS	M	Mgt	7.8.11
Event Handling Functions			
HBA_GetEventBuffer	M	Mgt	7.9.2
HBA_RegisterForAdapterAddEvents	M	O	7.9.4
HBA_RegisterForAdapterEvents	M	O	7.9.5
HBA_RegisterForAdapterPortEvents	M	Mgt	7.9.6
HBA_RegisterForAdapterPortStatEvents	M	O	7.9.7
HBA_RegisterForTargetEvents	M	O	7.9.8
HBA_RegisterForLinkEvents	M	Mgt	7.9.9
HBA_RemoveCallback	M	Mgt	7.9.10

Key:
M designates a function that is mandatory.
Mgt designates a function that is management-mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Table A.2 — Function Requirements for Systems Supporting SB

Function	For HBA API library [a]	For HBA specific libraries [a]	Reference
SB Information Functions			
HBA_GetSBTargetMapping	M	O	<by editor>
HBA_GetSBStatistics	M	O	<by editor>
SB Disk Device Information Functions			
HBA_SBDskGetCapacity	M	O	<by editor>

Key:
M designates a function that is mandatory.
Mgt designates a function that is management-mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

[a] The functions in this table shall be not allowed for all implementations of FC-HBA compliant software on systems that do not support SB devices, and shall be required as specified in the table for all implementations of FC-HBA compliant software on systems that support SB devices.

A.3 HBA Attributes

Table A.3 specifies the requirements for implementing HBA attributes for software that is compliant with FC-HBA and software that is compliant with FC-HBA Management Interoperability Extensions.

Table A.3 — HBA Attributes

Attribute Name	Requirement	Value indicating unspecified	Reference
Manufacturer	M	not applicable	6.3.2.2
SerialNumber	M	not applicable	6.3.2.3
Model	M	not applicable	6.3.2.4
ModelDescription	O	Null string	6.3.2.5
NodeWWN	O	Eight null bytes	6.3.2.6
NodeSymbolicName	O	Null string	6.3.2.7
HardwareVersion	O	Null string	6.3.2.8
DriverVersion	O	Null string	6.3.2.9
OptionROMVersion	O	Null string	6.3.2.10
FirmwareVersion	O	Null string	6.3.2.11
VendorSpecificID	O	zero	6.3.2.12
NumberOfPorts	M	not applicable	6.3.2.13
DriverName	O	Null string	6.3.2.14

Key:
M designates a function that is mandatory.
Mgt designates a function that is management-mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

A.4 FC_Port Attributes

Table A.4 specifies the the requirements for implementing FC_Port Attributes for software that is compliant with FC-HBA and software that is compliant with FC-HBA Management Interoperability Extensions. Different requirements are specified for attributes of local HBA end ports than for discovered FC_Ports.

Table A.4 — FC_Port Attributes

Attribute Name	For local end ports	for discovered FC_Ports	Value indicating unspecified	Reference
NodeWWN	M	O	eight null bytes	6.4.2.2
PortWWN	M	M	not applicable	6.4.2.3
PortSymbolicName	O	O	null string	6.4.2.4
PortFcId	M	M	not applicable	6.4.2.5
PortType	M	O	HBA_PORTTYPE_UNKNOWN	6.4.2.6
PortState	M	O	HBA_PORTSTATE_UNKNOWN	6.4.2.7
PortSupportedClassofService	M	O	zero	6.4.2.8
PortSupportedFc4ypes	M	O	32 null bytes	6.4.2.9
PortActiveFc4ypes	M	M	not applicable	6.4.2.10
PortSupportedSpeed	M	O	HBA_PORTSPEED_UNKNOWN	6.4.2.11
PortSpeed	M	O	HBA_PORTSPEED_UNKNOWN	6.4.2.12
PortMaxFrameSize	M	O	zero	6.4.2.13
OSDeviceName	M	O	null string	6.4.2.14
NumberofDiscoveredPorts	M	N	zero	6.4.2.15
FabricName	M	O	zero	6.4.2.16

Key:
M designates a function that is mandatory.
Mgt designates a function that is management-mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

A.5 End Port Statistics

Table A.5 specifies the requirements for implementing end port statistics for software that is compliant with FC-HBA and software that is compliant with FC-HBA Management Interoperability Extensions.

For any end port statistic, the value indicating unspecified shall be negative one (-1).

Table A.5 — End Port Statistics

Attribute Name	Requirement	Reference
SecondsSinceLastReset	O	6.5.2.2
TxFrames	O	6.5.2.3
RxFrames	O	6.5.2.4
TxWords	O	6.5.2.5
RxWords	O	6.5.2.6
LIPCount	O	6.5.2.7
NOSCount	O	6.5.2.8
ErrorFrames	O	6.5.2.9
DumpedFrames	O	6.5.2.10
LinkFailureCount	M	6.5.2.11
LossOfSyncCount	M	6.5.2.12
LossOfSignalCount	M	6.5.2.13
PrimitiveSeqProtocolErrCount	M	6.5.2.14
InvalidTxWordCount	M	6.5.2.15
Invalid CRC Count	M	6.5.2.16
InputRequests	O	6.5.2.17
OutputRequests	O	6.5.2.18
ControlRequests	O	6.5.2.19
InputMegabytes	O	6.5.2.20
OutputMegabytes	O	6.5.2.21

Key:
M designates a function that is mandatory.
Mgt designates a function that is management-mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

A.6 SB Statistics

Table A.6 specifies the requirements for implementing SB statisticsfor software that is compliant with FC-HBA on a system that supports SB and software that is compliant with FC-HBA Management Interoperability Extensions on a system that supports SB. This table shall not apply to compliant software on systems that do not support SB.

For any SB statistic, the value indicating unspecified shall be negative one (-1).

Table A.6 — SB Statistics

Attribute Name	**Requirement**	**Reference**
SSCHRSCHCount	O	<by editor>
SampleCount	O	<by editor>
DeviceConnectTime	O	<by editor>
FunctionPendingTime	O	<by editor>
DeviceDisconnectTime	O	<by editor>
ControlUnitQueuingTime	O	<by editor>
DeviceActiveOnlyTime	O	<by editor>
Key: M designates a function that is mandatory. Mgt designates a function that is management-mandatory. O designates a function that is optional. N indicates a function that is not allowed.		

A.7 FC-3 Management Attributes

Table A.7 specifies the requirements for implementing FC-3 management attributes for software that is compliant with FC-HBA and software that is compliant with FC-HBA Management Interoperability Extensions. Different requirements are specified for return of attributes than for setting them. 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 for setting.

Table A.7 — FC-3 Management Attributes

Attribute Name	Getting [a]	Setting [a]	Reference
WWN [a]	M	N	6.8.3.2
unittype	M	O	6.8.3.3
PortId	M	O	6.8.3.4
NumberOfAttachedNodes	M	O	6.8.3.5
IPVersion	M	O	6.8.3.6
UDPPort	M	O	6.8.3.7
IPAddress	M	O	6.8.3.8
TopologyDiscoveryFlags	M	O	6.8.3.9

Key:
M designates a function that is mandatory.
Mgt designates a function that is management-mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

[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.8 Library Attributes

Table A.5 specifies the requirements for implementing library attributes for software that is compliant with FC-HBA and software that is compliant with FC-HBA Management Interoperability Extensions.

For any Library Attribute, the value indicating unspecified shall be negative one (-1).

Table A.8 — Library Attributes

Attribute Name	Requirement	Reference
final	M	6.11.2.2
LibPath	M	6.11.2.3
VName	M	6.11.2.4
VVersion	M	6.11.2.5
tm_sec	O	6.11.1, 6.11.2.6
tm_min	O	6.11.1, 6.11.2.6
tm_hour	O	6.11.1, 6.11.2.6
tm_mday	M	6.11.1, 6.11.2.6
tm_mon	M	6.11.1, 6.11.2.6
tm_year	M	6.11.1, 6.11.2.6
tm_wday	O	6.11.1, 6.11.2.6
tm_yday	O	6.11.1, 6.11.2.6
tm_isdst	O	6.11.1, 6.11.2.6

Key:
M designates a function that is mandatory.
Mgt designates a function that is management-mandatory.
O designates a function that is optional.
N indicates a function that is not allowed.

Annex B
(normative)

Mapping FC-HBA to InfiniBand^{tm}

B.1 Structure and Concepts

B.1.1 Overview

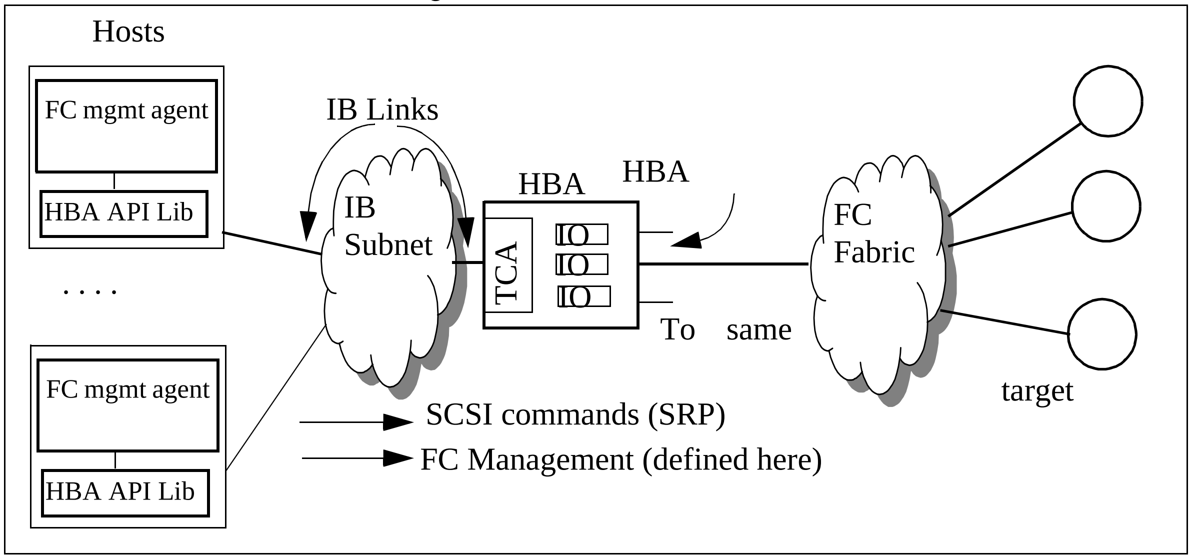
Improvements in the throughput and processing power of FC Host Bus Adapters (HBAs) have made it necessary to develop corresponding enhancements in host connection capabilities. To meet this need, HBAs that attach to the host with the InfiniBand Transport^{tm} (IB) are being offered. These HBAs provide the necessary enhancements in host connection performance, and they also allow multiple hosts to share usage of the same HBA, thereby allowing for greater utilization of the HBA capabilities. In order to minimize the software impact of using these new HBAs, it is advantageous to continue to provide the HBA API for them.

In order to implement the HBA API, the host needs to communicate with the HBA in order to send and receive various SCSI commands, send and receive FC ELSs and FC-CT commands, and obtain various parameters and performance statistics about fibre channel ports (FC_Ports) on the HBA. The SRP mapping of SCSI onto IB (see SRP) provides the host with the means to send and receive SCSI commands over IB, however, there is no standard way for the host to perform FC management functions such as sending FC ELSs and FC-CT commands, and obtaining parameters and performance statistics for HBA FC_Ports. Therefore, a standard protocol for transmitting these non-SCSI FC management functions over IB is needed to minimize the software effort required to implement the HBA API for IB-attached HBAs from multiple vendors. To avoid having to support multiple vendor-specific methods of sending and receiving FC management functions, a single standard method is needed.

This annex defines a protocol, referred to as the FC management service, for transferring FC management functions between FC management initiators in a host, and an HBA that is attached to the host by IB. The HBA API library (i.e., the library) acts as the FC management initiator; however, other applications (e.g., independent operating systems) may also act as FC management initiators. This annex also defines the methods by which the HBA API library obtains other information regarding IB-attached HBAs (e.g., the set of accessible HBAs or HBA attributes).

Figure B.1 shows how an HBA API library uses this annex. The HBA in figure B.1 consists of an IB Target Channel Adapter (TCA), one or more HBA FC_Ports, and one or more I/O Controllers (IOCs) (see B.1.2.5). The FC management agents in figure B.1 are applications that make calls to the HBA API library. The HBA API library performs the methods and protocol defined in this annex on behalf of the FC management agents when communicating with the IB-attached HBA in order to support the HBA API.

Figure B.1 — HBA with IB Host Attachment



This annex includes:

a) a description of the method for discovering the accessible HBAs in an IB subnet;
b) a description of the methods for discovering HBA attributes;
c) a definition of an FC management service providing the following functions;
   A) establishing a communication path between each FC management initiator and each HBA;
   B) sending FC ELSs and FC-CT commands from the FC management initiator to a specific FC_Port on the HBA and receiving responses (see subclause 7.8);
   C) sending RSCN or RLIR ELS requests received from a specific port on the HBA to the FC management initiator;
   D) sending HBA FC_Port parameters and statistics to FC management initiators;
   E) associating FC management initiators with corresponding FC parameters such as FC_Port Name_Identifier, FC Node_Name, and FC-3 management attributes for each HBA FC_Port (see subclause 6.8.3 for a definition of FC-3 management attributes); and
   F) associating SRP initiators with corresponding FC parameters such as FC_Port Name_Identifiers and Node_Names, and node identification data; and
d) a definition of the rules for choosing SRP target port identifiers so that each SRP target port identifier corresponds to a specific target FC_Port.

B.1.2 FC management Service Operations

B.1.2.1 Host to HBA Communication

Communication of FC management information between the host and the HBA shall use the FC management service. Information shall be transferred over one or more IB reliable connections. The FC management initiator shall initiate each connection using IB Communications Management, and shall act as the client. The HBA shall act as the server. When the FC management service is used to support the HBA API, the library shall act as an FC management initiator.

B.1.2.2 Service Name for the FC Management Service

The service name for the FC management service shall be a 64-byte, null-terminated string of UTF-8 encoded characters of the following format:

‘FCMGT.T11’.

B.1.2.3 Registration of the FC Management Service Name by HBAs

TCAs that support the FC management service shall send a SubnAdminSet(ServiceRecord) request to the IB Subnet Administrator (SA) to register as an FC management service provider prior to performing FC management service operations.

If a TCA supports the FC management service, then it shall support all the requirements of this annex. TCAs that support the FC management service are referred to as HBAs.

B.1.2.4 Discovery of the FC Management Service

To discover the accessible IB-attached HBAs, the FC management initiator determines the set of IB Target Channel Adapters (TCAs) that support the FC management service. One method of obtaining this information is to use the IB Administration Query Subsystem by sending an IB SubnAdminGet(ServiceRecord) request to the subnet administrator. The request specifies that the subnet administrator is to return all the ServiceRecord attributes containing the FC management service name (see IBA).

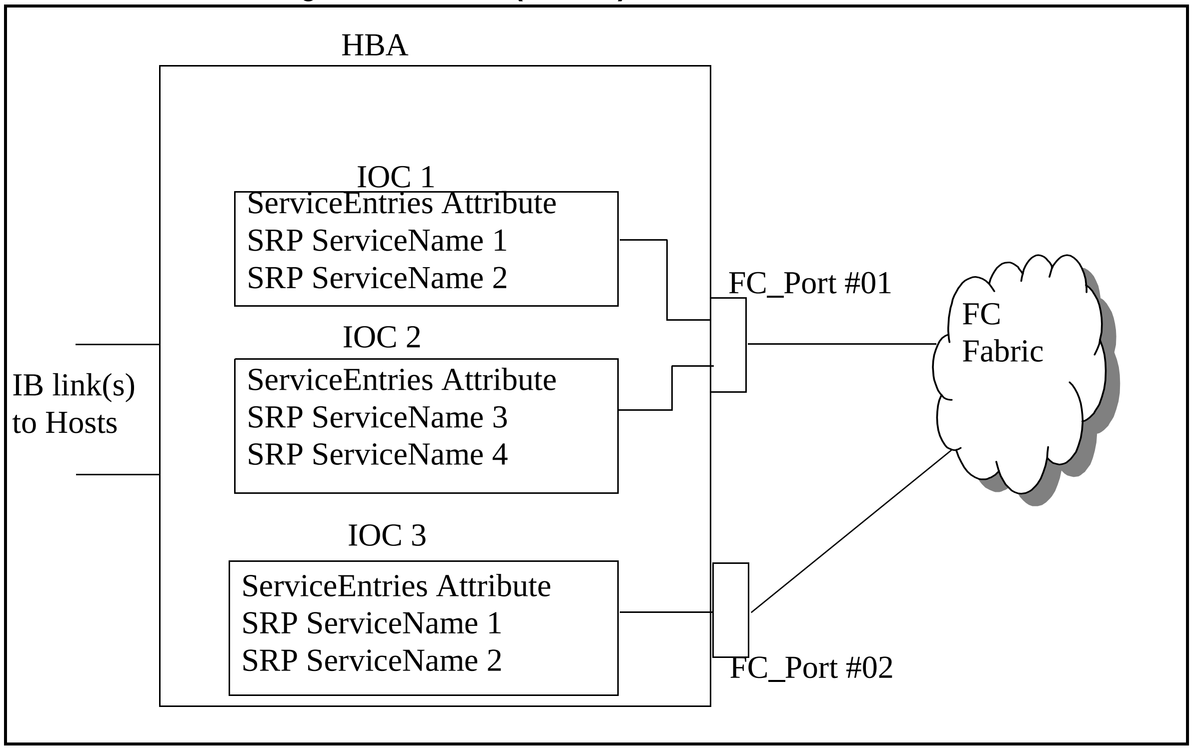
B.1.2.5 HBA Components

HBAs shall contain at least one I/O Controller (IOC) (see IBA for additional information regarding IOCs). Each IOC shall support the SRP protocol (see SRP), and shall provide at least one SRP Service Name in its ServiceEntries attribute (see B.1.3).

An IOC shall not provide access to more than one HBA FC_Port. Multiple IOCs may provide access to a single HBA FC_Port, however, no two IOCs providing access to a single HBA FC_Port shall support an SRP ServiceName that corresponds to the same target end port (see B.1.3.3).

Since IOCs do not provide access to more than one HBA FC_Port, all SRP ServiceNames supported by that IOC correspond to FC target ports that are accessed through that HBA FC_Port (see figure B.2).

Figure B.2 — HBA (I/O Unit) with Three IOCs and Two FC_Ports



B.1.2.6 FC Management Service Connections

B.1.2.6.1 Establishing a Connection

The FC management initiator shall establish one or more IB reliable connections to the FC management service for each HBA (see IBA for information about IB reliable connections). The FC management initiator shall initiate an IB reliable connection by sending an IB Connection Management (CM) request. The private data field of the CM:request shall contain the Establish_Channel request (see B.2.2.)

After a connection is established, the FC management initiator may obtain the FC_Port Attributes for each HBA FC_Port by sending a Get_Port_Data request (see B.2.4). The FC management initiator may also transfer specific node identification data or request the assignment of a unique N_Port_ID for an HBA FC_Port by sending a Set_Port_Data request (see B.2.5). The Set_Port_data request provides the N_Port_Name and Node_Name, specific node identification data, and other data pertaining to a specific HBA FC_Port. If assignment of a unique N_Port_ID is not indicated, the HBA shall ignore the N_Port_Name and Node_Name in the Set_Port_Data request; if specific node identification data is provided, the HBA shall apply it to the FLOGI-assigned N_Port_ID of the HBA FC_Port.

When an HBA that supports N_Port_ID virtualization receives a Set_Port_Data request for an HBA FC_Port, it determines if it has already obtained an N_Port_ID corresponding to the Port_Name in the Set_Port_Data request. If no N_Port_ID has been obtained, the HBA shall obtain a unique N_Port_ID; if an N_Port_ID has already been obtained, the HBA shall not obtain another N_Port_ID. If an N_Port_ID is obtained, the HBA shall use the FC_Port Name_Identifier and FC Node_Name provided in the Set_Port_Data request when obtaining the N_Port_ID. Subsequently, when the HBA performs FC management functions on the HBA FC_Port on behalf of the FC management initiator, it shall use the N_Port_ID assigned to the FC management initiator.

If assignment of a unique N_Port_ID is not indicated in the Set_Port_Data request, or if the HBA FC_Port does not support N_Port_ID virtualization, then the HBA shall not obtain a unique N_Port_ID for the FC management initiator. When performing FC management functions on behalf of the FC management initiator, the HBA shall use its FLOGI-assigned N_Port_ID.

If the HBA successfully performs the Set_Port_Data request, including the acquisition of a unique N_Port_ID for the FC_Port, if required, it shall send a Set_Port_Data response. If the HBA does not successfully perform the Set_Port_Data request, it shall reject the request.

N_Port login shall not be performed as part of establishing an FC management connection or performance of the Set_Port_Data request (see B.1.3.4.1 and B.2.8.1 for information about N_Port login).

NOTE 19 The FDISC ELS may be used to obtain a unique N_Port_ID (see FC-FS).

B.1.2.6.2 Releasing a Connection

When a connection is to be released, the CM:DREQ message shall be sent. The FC management initiator may release the connection at any time. The HBA shall not release the connection.

Prior to accepting a CM:DREQ from the FC management initiator, HBAs that support N_Port_ID virtualization shall determine, for each HBA FC_Port, if the N_Port_ID corresponding to the FC management initiator is currently logged in with any target FC_Ports. If that N_Port_ID is currently logged in with one or more target FC_Ports and that N_Port_ID does not also correspond to any SRP initiators, then explicit N_Port logout shall be performed with all target FC_Ports currently logged in with that N_Port_ID.

If the N_Port_ID corresponding to the FC management initiator is currently logged in with target FC_Ports and that N_Port_ID also corresponds to one or more SRP initiators, then for each target FC_Port, the HBA shall:

a) not perform N_Port logout with the target FC_Port if an SRP login exists between one of the SRP initiators and the SRP target port identifier corresponding to the target FC_Port; or
b) complete any outstanding ELS and FC-CT requests with the target FC_Port, and perform explicit N_Port logout with the target FC_Port, if no SRP logins exist between any SRP initiator and the SRP target port identifier corresponding to the target FC_Port.

If the N_Port_ID corresponding to the FC management initiator does not correspond to an SRP initiator or another FC management initiator, and if no N_Port logins remain for the N_Port_ID after all required N_Port logouts have been performed for a given HBA FC_Port, then the HBA shall send a LOGO ELS to the fabric to release the N_Port_ID of the FC management initiator.

B.1.2.7 Information Units

Information sent between the FC management initiator and the HBA on an IB reliable connection shall be contained in Information Units (IUs). Each IU is classified as either a request or a response. Requests shall be used to establish a reliable connection, to convey ELS or FC-CT commands, or to convey other parameters. Responses shall be used to convey the corresponding responses and provide any information that was requested (see B.2).

B.1.2.8 Asynchronous Event Notification

Asynchronous events shall include the receipt of incoming RSCN and RLIR ELSs, and other events detected by the HBA. FC management initiators shall indicate whether or not they require notification of asynchronous events by sending a Set_Port_Data request (see B.2.5). For those FC management initiators that have indicated that they require notification of asynchronous events, the HBA shall send an AEN request upon receipt of the RSCN and LIRR ELSs, and upon the occurrence of HBA-detected events (see B.2.9.1).

B.1.2.9 HCA Receive Buffer Management

HCA receive buffer management restricts the number of outstanding (i.e., unacknowledged) AEN requests sent by the HBA on a given connection. In the Establish_Connection request, the FC management initiator shall indicate the maximum number of outstanding AEN requests that the HBA shall be allowed to send on the connection (see B.2.2.1).

All HCA buffers used to receive IUs from the HBA shall be at least 1 042 bytes.

B.1.2.10 HBA Receive Buffer Management

HBA receive buffer management restricts the number of outstanding (i.e., unacknowledged) FC management service requests sent by the HCA on a given connection. In the Establish_Connection response, the HBA shall indicate the maximum number of outstanding FC management requests that the HCA shall be allowed to send on the connection (see B.2.2.2).

All HBA buffers used to receive FC management requests shall be at least 64 bytes.

B.1.2.11 Data Buffers

B.1.2.11.1 Memory Descriptors

A memory descriptor is a 16-byte structure that identifies an IB memory segment in the host (see IBA for information regarding IB memory segments). The format of the memory descriptor is shown in table B.1:.

Table B.1 — Memory Descriptor

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
...				VIRTUAL ADDRESS				
7								(LSB)
8	(MSB)							
...				REMOTE KEY				
11								(LSB)
12	(MSB)							
				DATA LENGTH				
15								(LSB)

The VIRTUAL ADDRESS field contains the IB virtual address of the memory segment. When accessing host memory, the HBA places this value in the VIRTUAL ADDRESS field of the Reliable Datagram Extended Transport header (see IBA).

The REMOTE KEY field contains the IB Remote Key that the HBA uses when accessing host memory (see IBA).

The DATA LENGTH field is an unsigned binary integer that specifies the length in bytes of the host memory segment.

B.1.2.11.2 Data Buffer Descriptors

A request may contain a data-out buffer descriptor, a data-in buffer descriptor, or both. If a data buffer descriptor defines a data-out buffer, the HBA shall only issue RDMA Read operations using the memory descriptor contained in the data buffer descriptor. If a data buffer descriptor defines a data-in buffer, the HBA shall only issue RDMA Write operations using the memory descriptor contained in the data buffer descriptor. There are two types of data buffer descriptors: direct and indirect. Both types of buffer descriptors contain a single memory descriptor as defined in B.1.2.11.1.

The memory descriptor in a direct buffer descriptor identifies the data buffer. The HBA shall use the contents of the DATA LENGTH field of the memory descriptor as the length of the data-out buffer or data-in buffer.

The memory descriptor in an indirect buffer descriptor specifies a memory segment containing an indirect table. An indirect table is a list of one or more memory descriptors. The memory segments specified by the memory descriptors in the indirect table form the data buffer. The value of the DATA LENGTH field of the memory descriptor represents the length, in bytes, of the indirect table, and is the number of memory descriptors in the indirect table multiplied by sixteen (the length, in bytes, of a memory descriptor).

The format of each data buffer descriptor is specified by a format code value in the FC management request. In an FC management request that contains both data-in and data-out buffer descriptors, there is no requirement that both buffer descriptors be of the same format.

The buffer descriptor format codes shall be as shown in table B.2.

Table B.2 — Data Buffer Descriptor Format Codes

Data Buffer Descriptor Format Code	Value
NO DATA BUFFER DESCRIPTOR PRESENT [a]	0h
DIRECT BUFFER DESCRIPTOR	1h
INDIRECT BUFFER DESCRIPTOR	2h

[a] When no data buffer descriptor is present, the HBA shall ignore the buffer descriptor in the FC management request.

B.1.3 FC Management Support for SCSI Operations

B.1.3.1 Host to HBA Communications

SRP initiators shall use the SRP protocol (see SRP) for transferring SCSI commands and data to the HBA. Hosts shall support at least one SRP initiator. HBAs shall support a unique SRP target port identifier for every target FC_Port that is accessible to the FLOGI-assigned N_Port_ID on each HBA FC_Port (see B.1.3.3).

B.1.3.2 Discovery of SRP Target Ports

SRP target ports supported by the HBA shall be discovered using the SRP target port discovery process defined in SRP (see SRP).

HBAs that support N_Port_ID virtualization have the ability to assign each SRP initiator with a unique N_Port_ID (see B.1.3.4). For these HBAs, an SRP initiator that has been assigned its own N_Port_ID might not be able to access all of the SRP target port IDs supported by the HBA due to FC zoning. If an SRP initiator attempts to perform SRP login with an SRP target port ID corresponding to a target end port to which the SRP initiator’s N_Port_ID does not have access, then the HBA shall reject the SRP login request (see B.1.3.4.1).

B.1.3.3 SRP Target Port IDs

In order implement the HBA API, SRP initiators require a direct communication path for sending SCSI commands to individual target FC_Ports. To provide this capability, HBAs shall choose SRP target port IDs so that there is a one-to-one correspondence between each SRP target port ID and each target FC_Port accessible to the HBA. The extension portion of each SRP target port ID shall be set to the target FC_Port Name_Identifier of the target FC_Port to which the SRP target port ID corresponds. The format of the SRP target port ID shall be as follows:

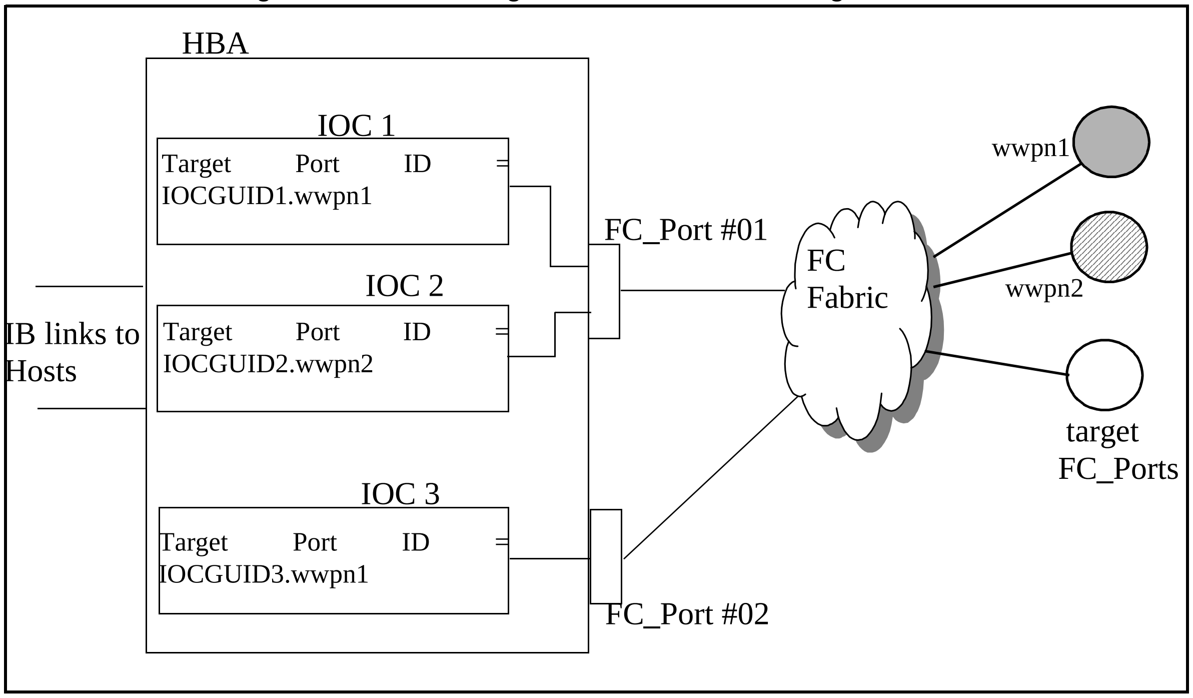
SRP Target Port ID = IOCGUID.wwpn,

where wwpn = target FC_Port Name_Identifier.

figure B.3 shows an example of SRP target port identifiers.

The library or operating system supporting an SRP initiator may determine the FC_Port Name_Identifier of the target FC_Port that is accessible on a connection from the SRP initiator to an SRP target port by examining the wwpn portion of the SRP target port ID (see SRP for additional information about SRP target port IDs).

Figure B.3 — SRP Target Port IDs and FC Management Service Names


HBA
IOC 1
Target Port ID = IOCGUID1.wwpn1
IOC 2
Target Port ID = IOCGUID2.wwpn2
IOC 3
Target Port ID = IOCGUID3.wwpn1
IB links to Hosts
FC_Port #01
FC_Port #02
FC Fabric
wwpn1
wwpn2
target FC_Ports

In figure B.3, when an SRP initiator forms a connection to SRP target port ID ‘IOCGUID1.wwpn1’, FC_Port #01 is being used since the response to the Get_Port_Data FC management service request indicated that FC_Port #01 was accessed by IOC1 (see B.2.4). When an SRP initiator forms a connection to SRP target port ID ‘IOCGUID2.wwpn2’, FC_Port #01 is also being used because the response to the Get_Port_Data FC management service request indicated that FC_Port #01 was also accessed by IOC2. When an SRP initiator makes a connection to SRP target port ID ‘IOCGUID3.wwpn1’, however, FC_Port #02 is being used since the response to the Get_Port_Data FC management service request indicated that FC_Port #02 was accessed by IOC3. Note that target FC_Port wwpn1 is not accessible using both IOC1 and IOC2 since both IOCs use the same HBA FC_Port (see B.1.2.5).

When more than 256 target FC_Ports may be accessed from a single HBA FC_Port, multiple IOCs shall be used to provide access to the HBA FC_Port since each IOC may provide access to only 256 SRP target port IDs. Since there does not need to be any relationship between IOCs reported to the host and physical IOCs on the HBA, a single physical IOC on the HBA may be reported as multiple IOCs to the host.

B.1.3.4 SRP Connections

B.1.3.4.1 Establishing an SRP Connection

SRP connections shall be established using SRP login as specified in SRP (see SRP). (i.e., one IB reliable connection shall be established between each SRP initiator and each SRP target port with which the SRP initiator communicates.) The SRP login request is carried in the private data field of the IB CM:REQ message.

When an HBA that does not support N_Port_ID virtualization receives an SRP login request, it determines if it is already logged in with the target FC_Port corresponding to the SRP target port ID in the SRP login request (see B.1.3.3). If the HBA is already logged in, it accepts the connection if resources are available. If it is not logged in, the HBA attempts to perform N_Port login with the target FC_Port corresponding to the SRP target port ID in the SRP login request. If N_Port login is successful and resources are available, the HBA accepts the SRP login request; if N_Port login is not successful or if resources are not available, it rejects the SRP login request.

When an HBA that supports N_Port_ID virtualization receives an SRP login request, it determines if it has completed a Bind_SRP_Initiator request for the SRP initiator. If no Bind_SRP_Initiator request for the SRP initiator has been completed, the HBA rejects the connection request. SRP Initiator binding is the process of pairing an SRP initiator port identifier with an FC_Port Name_Identifier, Node_Name, and specific node descriptor that the HBA uses on behalf of that SRP initiator (see B.2.6 for information about the Bind_SRP_Initiator function).

If a Bind_SRP_Initiator request that provided the FC_Port Name_Identifier and Node_Name for the SRP initiator has been completed, then the HBA determines if it has already obtained an N_Port_ID corresponding to the SRP initiator. If no N_Port_ID has been obtained, the HBA shall obtain an N_Port_ID for the SRP initiator. The FC_Port Name_Identifier and Node_Name that were bound to the SRP initiator in the Bind_SRP_Initiator request shall be used when obtaining the N_Port_ID.

If an N_Port_ID for the SRP initiator is obtained, or if the requesting SRP initiator has already been assigned an N_Port_ID, then the HBA determines if the N_Port_ID corresponding to the SRP initiator is currently logged in with the target FC_Port corresponding to the SRP target port identifier in the SRP login request. If the N_Port_ID corresponding to the SRP initiator is currently logged in, then the HBA accepts to the connection request, if resources are available. If the N_Port_ID corresponding to the SRP initiator is not currently logged in, then the HBA shall query the nameserver using the N_Port_ID corresponding to the SRP initiator to determine if the target FC_Port is accessible to the SRP initiator. If the target FC_Port is accessible to the SRP initiator, then the HBA shall complete N_Port login with the target FC_Port using the N_Port_ID corresponding to the SRP initiator, and shall accept the SRP login request. If the destination target FC_Port is not accessible to the N_Port_ID corresponding to the SRP initiator, then the HBA shall reject the SRP login without attempting N_Port login.

In order to minimize unsuccessful SRP login attempts due to zoning restrictions, an FC management connection may be established, and the same FC_Port Name_Identifier and Node_Name as the SRP initiator may be provided in the Set_Port_Data request. This causes the FC management initiator and SRP initiator to share an N_Port_ID. Consequently, when the FC management initiator sends a Send_Passthru request to query the nameserver for the set of accessible target FC_Ports, the nameserver responds with the set of target FC_Port Name_Identifiers that are accessible to the N_Port_ID corresponding to the SRP initiator. The accessible SRP target port identifiers may then be generated as specified in B.1.3.3.

If a Bind__SRP_Initiator request specifying that the FLOGI-assigned FC_Port Name_Identifier is to be used on behalf of the SRP initiator has been completed for the requesting SRP initiator, then the HBA determines if its FLOGI-assigned N_Port_ID is already logged in with the target FC_Port corresponding to the SRP target port ID in the SRP login request (see B.1.3.3). If the HBA’s FLOGI-assigned N_Port_ID is already logged in with the target FC_Port, shall accept the connection if resources are available. If the HBA’s FLOGI-assigned N_Port_ID is not logged in, the HBA attempts to perform N_Port login with the target FC_Port corresponding to the SRP target port ID in the SRP login request. If N_Port login successful and resources are available, the HBA accepts the SRP login request; if N_Port login is not successful or if resources are not available, it rejects the SRP login request.

After SRP login has been successfully completed, the HBA shall convert subsequent SRP commands and SRP task management requests initiated by the SRP initiator into FCP commands and task management operations with the target FC_Port corresponding to the SRP target port identifier. The HBA shall also transfer the related data between the SRP initiator and target FC_Port, and shall send ending status received from the target FC_Port to the SRP initiator.

B.1.3.4.2 Releasing an SRP Connection

SRP connections are released as specified by SRP (see SRP).

Prior to accepting a CM:DREQ from an SRP initiator, HBAs that support N_Port_ID virtualization shall determine if the N_Port_ID corresponding to the SRP initiator also corresponds to other SRP initiators. If the N_Port_ID corresponding to the SRP initiator does not also correspond to any other SRP initiator, then explicit N_Port login with the target FC_Port shall be performed.

If the N_Port_ID corresponding to the SRP initiator also corresponds to one or more other SRP initiators the HBA shall:

a) not perform N_Port logout with the target FC_Port if an SRP login exists between one of the SRP initiators and the SRP target port identifier corresponding to the target FC_Port; and
b) complete any outstanding ELS and FC-CT requests with the target FC_Port, and perform explicit N_Port logout with the target FC_Port if no SRP logins exist between any SRP initiator and the SRP target port identifier corresponding to the target FC_Port.

If N_Port logout with the target FC_Port is performed and no N_Port logins remain for the N_Port_ID corresponding to the SRP initiator, and if the N_Port_ID corresponding to the SRP initiator does not also correspond to an FC management initiator or SRP initiator, then the HBA shall send a LOGO ELS to the fabric to release the N_Port_ID of the SRP initiator, provided that the KEEPID bit in the Bind_SRP_Initiator request was set to zero (see B.2.6.1).

B.2 Information Units

B.2.1 Summary

Communication between FC management initiators and the HBA is carried in FC management requests and responses. Each request and response message shall contain a single information unit (IU), that specifies the meaning of the request or response, and transfers the applicable parameters.

Byte 0 of each IU shall contain a TYPE code. The TYPE code value shall uniquely identify the IU. The length of an IU shall be indicated by its TYPE code and selected fields within the IU. If an IU is received with an invalid TYPE code, or whose length is incorrect for the IU, the recipient shall reject the request (see B.2.6).

Bytes 8-15 of each IU contain a TAG value, that provides a mechanism for matching requests with their corresponding responses. Each request IU shall contain a TAG value that is unique among all outstanding requests sent on an IB reliable connection. Each response shall contain a copy of the TAG value from the corresponding request. Responders are not required to check whether the TAG values of outstanding requests are unique; if the TAG values are not unique, responder behavior is unpredictable.

B.2.2 Establish_Connection

B.2.2.1 Establish_Connection request

The Establish_Connection request shall request the establishment of an FC management connection to an HBA. The Establish_Connection request shall be carried in the private data field of the IB CM:Request message that is used to establish the IB reliable connection with the HBA (see B.1.2.6.1). See table B.3 for the format of the Establish_Connection request.

Table B.3 — Establish_Connection request

Bit Byte	7	6	5	4	3	2	1	0
0	TYPE (00h)							
1	INBOUND REQUEST LIMIT							
2								
...				reserved				
7								
8	(MSB)							
...				TAG				
15								(LSB)

The INBOUND REQUEST LIMIT field shall be an unsigned binary integer specifying the maximum number of outstanding AEN requests for which no response has been received that the HBA is allowed to send on the FC management connection. The value of the INBOUND REQUEST LIMIT shall be between 1 and 255.

B.2.2.2 Establish_Connection response

The Establish_Connection response shall indicate successful establishment of the IB connection.The Establish_Connection response shall be carried in the private data field of the CM:Response message. See table B.4 for the format of the Establish_Connection response.

Table B.4 — Establish_Connection response

Bit Byte	7	6	5	4	3	2	1	0
0	TYPE (A0h)							
1	OUTBOUND REQUEST LIMIT							
2								
...				reserved				
7								
8	(MSB)							
...				tag				
15								(LSB)

The OUTBOUND REQUEST LIMIT field shall be an unsigned binary integer specifying the maximum number of outstanding FC management service requests for which no response has been received that the FC management initiator is allowed to send on the FC management connection. The value of the OUTBOUND REQUEST LIMIT shall be between 64 and 255.

B.2.2.3 Establish_Connection_REJ response

The Establish_Connection_REJ response shall indicate that the HBA is unable to establish an IB connection. The Establish_Connection_REJ response shall be carried in the private data field of the CM:Reject message. See table B.5 for the format of the Establish_Connection_REJ response.

Table B.5 — Establish_Connection_REJ response

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (B0h)							
1	REASON CODE							
2	reserved							
...								
7								
8	(MSB)							
...	TAG							
15								(LSB)

The REASON CODE shall indicate the reason that the IB reliable connection was not established. The codes and their meanings are defined in table B.6.

Table B.6 — Establish_Connection_REJ response reason codes

Reason Code	Description
01h	Unable to establish IB connection, no reason specified
02h	Reserved
03h	Busy
all other values	Reserved

B.2.3 Get_Adapter_Attributes

B.2.3.1 Get_Adatper_Attributes request

The Get_Adatper_Attributes request shall be used to obtain the HBA attributes. See table B.7 for the format of the Get_Adapter_Attributes request.

Table B.7 — Get_Adapter_Attributes request

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (01h)							
1	reserved							
...								
6								
7						DATA-IN BUFFMT		
8	(MSB) TAG							
...								
15								(LSB)
16	(MSB) DATA-IN BUFFER DESCRIPTOR							
31								(LSB)

The DATA-IN BUFFMT field shall specify the format of the DATA-IN BUFFER DESCRIPTOR (see B.1.2.11.2).

The DATA-IN BUFFER DESCRIPTOR shall contain a buffer descriptor for a data buffer that the HBA shall use to store the adapter attributes (see B.2.3.2).

B.2.3.2 Get_Adapter_Attributes response

The Get_Adapter_Attributes response shall be used to indicate completion of the Get_Adapter_Attributes request. See table B.8 for the format of the Get_Adatper_Attributes response..

Table B.8 — Get_Adapter_Attributes response

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (A1h)							
1	REASON CODE							
2	reserved							
3								
4	(MSB)							
...	AVAILABLE DATA							
7								(LSB)
8	(MSB)							
...	TAG							
15								(LSB)
16	(MSB)							
...	VENDOR SPECIFIC ID							
19								(LSB)

The REASON CODE shall indicate the completion status of the Get_Adapter_Attributes request. See table B.9 for a definition of the codes and their meanings.

The value of AVAILABLE DATA shall be the size in bytes of the list of all adapter attributes available from the HBA, regardless of the size of the data-in buffer.

Table B.9 — Get_Adapter_Attributes response reason codes

Reason Code	Description
00h	Request successfully completed
01 - 02h	Reserved
03h	Busy. The HBA shall not access the data-in buffer.
04h-08h	Reserved
09h	The size of available data exceeds the size of the data-in buffer.
all other values	Reserved

The vendor specific ID shall be a 32-bit vendor-specific binary integer identifying the adapter.

The data-in buffer shall contain the adapter attributes. The adapter attributes shall be as shown in table B.10 (see subclause 6.3.2 of this standard for a definition of each adapter attribute). Each attribute shall be a null-terminated ASCII string. The next attribute shall immediately follow the terminating ASCII null character of the previous attribute. If the size of the entire attribute list available from the HBA exceeds the size of the data-in buffer, the attribute list shall be truncated after the last attribute that completely fits in the data-in buffer, and the remaining bytes of the data-in buffer, if any, shall be set to zero.

Table B.10 — Adapter Attributes

Attribute	Maximum Number of Characters
Manufacturer	64
SerialNumber	64
Model	256
ModelDescription	256
NodeSymbolicName	256
HardwareVersion	256
DriverVersion	256
OptionROMVersion	256
FirmwareVersion	256
DriverName	256

B.2.4 Get_Port_Data

B.2.4.1 Get_Port_Data request

The Get_Port_Data request shall request information regarding the HBA FC_Ports. See table B.11 for the format of the Get_Port_Data request.

Table B.11 — Get_Port_Data request

Bit Byte	7	6	5	4	3	2	1	0
0	TYPE (02h)							
1	reserved							
...								
6								
						DATA-IN BUFFMT		
8	(MSB)							
...	TAG							
15								(LSB)
16	(MSB)							
...	DATA-IN BUFFER DESCRIPTOR							
31								(LSB)

The DATA-IN BUFFMT field shall specify the format of the DATA-IN BUFFER DESCRIPTOR (see B.1.2.11.2).

The DATA-IN BUFFER DESCRIPTOR shall contain a buffer descriptor for a data buffer that the HBA shall use to store the port attributes (see B.2.4.2).

B.2.4.2 Get_Port_Data response

The Get_Port_Data response shall be used to transfer information regarding HBA FC_Ports. See table B.12 for the format of the Get_Port_Data response.

Table B.12 — Get_Port_Data response

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (A2h)							
1	REASON CODE							
2	reserved							
3								
4	(MSB)							
...	AVAILABLE DATA							
7								(LSB)
8	(MSB)							
...	TAG							
15								(LSB)

The REASON CODE shall indicate the completion status of the Get_Port_Data request. See table B.13 for a definition of the codes and their meanings.

The value of AVAILABLE DATA shall be the size in bytes of the list of all port descriptors available from the HBA, regardless of the size of the data-in buffer.

Table B.13 — Get_Port_Data response reason codes

Reason Code	Description
00h	Request successfully completed
01 - 02h	Reserved
03h	Busy. The HBA shall not access the data-in buffer.
04h-08h	Reserved
09h	The size of available data exceeds the size of the data-in buffer.
all other values	Reserved

The data-in buffer shall be used contain the port descriptors. The port descriptors shall contain the information regarding all HBA FC_Ports. There shall be one port descriptor for each HBA FC_Port, unless the available data exceeds the size of the data-in buffer. If the size of the entire port descriptor list available from the HBA exceeds the size of the data-in buffer, the port descriptor list shall be truncated after the last port descriptor that completely fits in the data-in buffer, and the remaining bytes of the data-in buffer, if any, shall be set to zero. See table B.14 for the format of the port descriptor.

Table B.14 — Port Descriptor

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)	PORT DESCRIPTOR LENGTH						
1								(LSB)
2	PHYSICAL PORT NUMBER							
3	reserved							
4	(MSB)	FC_PORT NAME_IDENTIFIER						
...								
11								(LSB)
12	(MSB)	FC NODE_NAME						
...								
19								(LSB)
20	(MSB)	N_PORT_ID						
21								
22								(LSB)
23	reserved							
24	(MSB)	FABRIC NAME						
...								
31								(LSB)
32	(MSB)	PORTTYPE						
...								
35								(LSB)
36	(MSB)	PORTSTATE						
...								
39								(LSB)
40	(MSB)	PORTSUPPORTEDCLASSOFSERVICE						
...								
43								(LSB)

Table B.14 — Port Descriptor

Bit Byte	7	6	5	4	3	2	1	0
44	(MSB)							
...				PORTSUPPORTEDFC4TYPES				
75								(LSB)
76	(MSB)							
...				PORTACTIVEFC4TYPES				
107								(LSB)
108	(MSB)							
...				PORTSUPPORTEDSPEED				
111								(LSB)
112	(MSB)							
...				PORTSPEED				
115								(LSB)
116	(MSB)							
...				PORTMAXFRAMESIZE				
119								(LSB)
120	(MSB)							
...				NUMBEROFDISCOVEREDPORTS				
123								(LSB)
124 - n				PORTSYMBOLICNAME				
n + 1				NUMBER OF IOCS				
n + 2								
...				IOCGUID LIST				
m								

The PORT DESCRIPTOR LENGTH field shall be an unsigned binary integer specifying the length in bytes of the port descriptor.

The PHYSICAL PORT NUMBER field shall be an unsigned 8-bit integer specifying the vendor-assigned physical port number. The number specified shall be the same as the physical port number provided in the General Topology Specific Node Identification data (see FC-FS).

The FC_PORT NAME_IDENTIFIER field shall be set to the Port_Name corresponding to the FLOGI-assigned N_Port_ID of the HBA FC_Port.

The NODE_NAME field shall be set to the Node_Name corresponding to the FLOGI-assigned N_Port_ID of the HBA FC_Port.

The N_PORT_ID field shall be set to the FLOGI-assigned N_Port_ID of the HBA FC_Port.

The FABRIC NAME field shall be set to the Fabric Name of the fabric that is attached to the HBA FC_Port. If the HBA FC_Port is not attached to a fabric, then the Fabric Name field shall be set to zero.

The PORTTYPE, PORTSTATE, PORTSUPPORTEDCLASSOFSERVICE, PORTSUPPORTEDFC4TYPES, PORTACTIVEFC4TYPES, PORTSUPPORTEDSPEED, PORTSPEED, PORTMAXFRAMESIZE, AND NUMBEROFDISCOVERED-PORTS fields are as defined in subclause 6.4.2 of this standard.

The PORTSYMBOLICNAME field shall be as defined in subclause 6.4.2 of this standard except that it shall terminate at the first word boundary (i.e., multiple of four bytes) following the zero byte that demarks the end of the actual name.

The NUMBER OF IOCS field shall be an unsigned binary integer indicating the number of IOCs that provide access to the HBA FC_Port (see B.1.2.5).

The IOCGUID LIST field shall contain a list of the IOCGUIDs of the IOCs that provide access to the HBA FC_Port (see B.1.2.5).

B.2.5 Set_Port_Data

B.2.5.1 Set_Port_Data request

The Port_Data request shall be used to request the assignment of a unique N_Port_ID for an HBA FC_Port to convey the parameters that shall apply to that N_Port_ID. See table B.15 for the format of the Set_Port_Data request.

Table B.15 — Set_Port_Data request

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (03h)							
1	PHYSICAL PORT NUMBER							
2	reserved							
...								
7				AEN	NPIV	DATA-OUT BUFFMT		
8	(MSB)							
...	TAG							
15								(LSB)
16	(MSB)							
...	FC_PORT NAME_IDENTIFIER							
23								(LSB)
24	(MSB)							
	FC NODE_NAME							
31								(LSB)
32	(MSB)							
	DATA-OUT BUFFER DESCRIPTOR							
47								(LSB)

The PHYSICAL PORT NUMBER field shall be set to the physical port number of the HBA FC_Port to which the Set_Port_Data request is applied (see B.2.4.2).

The DATA-OUT BUFFMT field shall specify the format of the DATA-OUT BUFFER DESCRIPTOR (see B.1.2.11.2).

The N_Port_ID Virtualization (NPIV) bit shall be set to zero if the FC management initiator does not require the assignment of a unique N_Port_ID for the HBA port. When the NPIV bit is set to zero, the HBA shall ignore the FC_PORT NAME_IDENTIFIER and FC NODE_NAME fields. When sending FC ELSs or FC-CT commands on behalf of the FC management initiator, the HBA shall use its FLOGI-assigned N_Port_ID.

The NPIV bit shall be set to one if the FC management initiator requires the assignment of a unique N_Port_ID. When the NPIV bit is set to one, the FC_PORT NAME_IDENTIFIER field shall contain a valid FC Port_Name, and the FC NODE_NAME field shall contain a valid FC Node_Name. When the NPIV bit is set to one, the HBA shall obtain a unique N_Port_ID for the FC management initiator unless it has already obtained an N_Port_ID corresponding to the Port_Name and Node_Name in the Set_Port_Data request. If an N_Port_ID is obtained, the HBA shall use the FC Port_Name and FC Node_Name provided by the FC management initiator when obtaining the N_Port_ID.

Set_Port_Data requests for a single HBA FC_Port may be sent on multiple FC management connections, and the same Port_Name and Node_Name may be provided in the Set_Port_Data requests. This occurs when a single FC management initiator establishes the connections, or when multiple FC management initiators establish the connections in order to share usage of the same N_Port_ID.

The asynchronous event notification (AEN) bit shall be set to one if the FC management initiator requires notification of asynchronous events for the HBA FC_Port; if the FC management initiator does not require notification of asynchronous events for the HBA FC_Port, the AEN bit shall be set to zero (see B.2.9).

When the DATA-OUT BUFFER DESCRIPTOR field contains a data out buffer descriptor, then the data-out buffer shall contain one or more data blocks containing the payload of an RNID accept ELS that the HBA is to use on behalf of the FC management initiator for the HBA FC_Port. If the HBA supports N_Port_ID virtualization and if the FC management initiator has requested the assignment of a unique N_Port_ID in the Set_Port_Data request, the HBA shall apply the node description data to the assigned N_Port_ID; if the FC management initiator did not request the assignment of a unique N_Port_ID in the Set_Port_Data request or if the HBA does not support N_Port_ID virtualization, then the HBA port shall apply the specific node identification data to its FLOGI-assigned N_Port_ID for the HBA FC_Port.

Each data block in the data-out buffer shall contain the entire payload of an the RNID accept ELS to be sent in response to an RNID ELS request of a different type.

If specific node identification data of the same type as requested in an incoming RNID request is not provided for the destination N_Port_ID, then the HBA shall respond as required by FC-FS when no specific node identification data is available. (i.e., it shall either reject the RNID ELS request, or shall accept the request and provide only the common node identification data.)

If specific Node Identification is not provided in the Set_Port_Data request, a Bind_SRP_Initiator request may later provide specific node identification for the same Port_Name and Node_Name provided in the Set_Port_Datal request. This may occur when an SRP Initiator shares an N_Port_ID with a FC management initiator (see B.2.6.1).

B.2.5.2 Set_Port_Data response

The Port_Data response shall indicate successful completion of the Set_Port_Data request. See table B.16 for the format of the Set_Port_Data response.

Table B.16 — Set_Port_Data response

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (A3h)							
1	reserved							
...								
7								
8	(MSB) TAG							
...								
15								(LSB)
16	reserved							NPIV
17	(MSB) N_PORT_ID							
19								(LSB)

The NPIV bit shall be set to one if the FC management initiator has requested that a unique N_Port_ID be assigned and the HBA and has obtained a unique N_Port_ID corresponding to the FC management initiator. The NPIV field shall be set to zero if the FC management initiator has not requested that a unique N_Port_ID be assigned.

If the NPIV bit is set to one, the N_Port_ID field shall be set to an N_Port_ID that is uniquely assigned to the FC management initiator. If the NPIV bit is set to zero, the N_Port_ID field shall be set to the FLOGI-assigned N_Port_ID.

B.2.5.3 Set_Port_Data_REJ Response

The Set_Port_Data_REJ response shall indicate that the Set_Port_Data request was not successfully performed. See table B.17 for the format of the Set_Port_Data_REJ response.

Table B.17 — Set_Port_Data_REJ response

Bit Byte	7	6	5	4	3	2	1	0
0	TYPE (B3h)							
1	REASON CODE							
2	reserved							
...								
7								
8	(MSB)							
...	TAG							
15								(LSB)

The REASON CODE field shall indicate the reason that the Set_Port_Data request was not performed. The codes and their meanings are defined in table B.18.

Table B.18 — Set_Port_Data_REJ response reason codes

Reason Code	Description
01h	One or more other Set_Port_Data or Bind_SRP_Initiator requests for this HBA FC_Port provided the same FC_Port Name_Identifier, but the Node_Name provided is not the same as the Node_Name provided in the other requests.
02h	Unable to obtain additional N_Port_IDs. This code shall be returned if the FC management initiator has requested the assignment of a unique N_Port_ID, but N_Port_ID virtualization not supported.
03h	Busy
04h	Insufficient resources to obtain an additional N_Port_ID. This code shall be returned if N_Port_ID virtualization is supported but there are no resources available, either in the HBA or in the Fabric, for obtaining an additional N_Port_ID.
05h	A Set_Port_Data request has already been completed on this connection for this HBA FC_Port, and a different FC_Port Name_Identifier or FC Node_Name was provided in the Set_Port_Data request. [a]
06h	An additional N_Port_ID was requested, but the HBA FC_Port is not attached to a Fabric.
all other values	Reserved

[a] When multiple Set_Port_Data requests for the same HBA FC_Port provide the same FC_Port Name_Identifier, the FC Node_Names provided shall also be the same. If SRP initiator identifiers have also been bound to the same FC_Port Name_Identifier, their FC Node_Names shall also be the same (see B.2.6).

B.2.6 Bind_SRP_Initiator

B.2.6.1 Bind_SRP_Initiator request

SRP Initiator binding is the process of pairing an SRP initiator port identifier with an FC Port_Name, FC Node_Name, and specific node identification data. After a bind is made for an SRP initiator, the HBA shall use the FC_Port Name_Identifier and FC Node_Name bound to the SRP initiator when obtaining an N_Port_ID corresponding to the SRP initiator, and when performing N_Port login on behalf of the SRP initiator. The HBA shall use the specific node identification data bound to an SRP initiator when responding to incoming RNID ELS requests for the N_Port_ID corresponding to the SRP initiator.

The Bind_SRP_Initiator request shall be used to bind an SRP initiator, (i.e., to send the SRP initiator port identifier, the FC Port_Name, Node_Name, and specific node identification data corresponding to the SRP initiator), to the HBA.

HBAs that support N_Port_ID virtualization shall support the Bind_SRP_Initiator request. An HBA that supports N_Port_ID virtualization shall not accept an SRP login request from an SRP initiator until the SRP initiator binding is established either by a Bind_SRP_Initiator request or other vendor specific means.

HBAs that do not support N_Port_ID virtualization shall reject the Bind_SRP_Initiator request.

See table B.19 for the format of the Bind_SRP_Initiator request.

Table B.19 — Bind_SRP_Initiator request

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (04h)							
1	PHYSICAL PORT NUMBER							
2	reserved					RMV	KEEPID	ASGN
3	reserved							
...								
6								
7						DATA-OUT BUFFMT		
8	(MSB)	TAG						
...								
15								(LSB)
16	(MSB)	SRP INITIATOR PORT IDENTIFIER						
31								(LSB)
32	(MSB)	FC_PORT NAME_IDENTIFIER						
39								(LSB)
40	(MSB)	NODE_NAME						
47								(LSB)
48	(MSB)	DATA-OUT BUFFER DESCRIPTOR						
63								(LSB)

The PHYSICAL PORT NUMBER field shall be set to the physical port number of the HBA FC_Port to which the Bind_SRP_Initiator request is applied (see B.2.4.2).

When the assign (ASGN) bit is set to zero, the HBA shall bind the SRP initiator to the FLOGI-assigned N_Port_ID. When sending SCSI commands on behalf of the SRP initiator, the HBA shall use its FLOGI-assigned N_Port_ID. When the ASGN field is set to zero, the FC_Port Name_Identifier, FC Node_Name, and Data-out Buffer descriptor fields shall be ignored by the HBA.

When the ASGN bit is set to one, the HBA shall bind the SRP initiator to the FC_Port Name_Identifier, FC Node_Name, and specific node identification data, if any provided in the Bind_SRP_Initiator Request. The HBA shall attempt to obtain an N_Port_ID for the SRP initiator prior to sending a Bind_SRP_Initiator response. If an N_Port_ID is obtained, the assigned N_Port_ID shall be indicated in the Bind_SRP_Initiator response; if an N_Port_ID cannot be obtained, the Bind_SRP_Initiator request shall be rejected.

When the KEEPID bit is set to zero, then the HBA shall send a LOGO ELS to the fabric to release the N_Port_ID corresponding to the SRP initiator after completion of an SRP logout by the SRP initiator, provided that no SRP logins remain for the SRP initiator, and if the N_Port_ID corresponding to the SRP initiator does not also correspond to another SRP initiator or an FC management initiator. When the KEEPID bit is set to one, then the HBA shall not release the N_Port_ID corresponding to the SRP initiator after completion of SRP logout by the SRP initiator.

The remove (RMV) bit shall be set to one in order to remove the bind for an SRP initiator. Prior to the time when a bind for an SRP initiator is removed, the SRP initiator shall complete SRP logout with all SRP targets with which it is logged in. When the RMV field is set to a one and the Bind_SRP_Initiator is accepted, the HBA shall remove the FC_Port Name_Identifier, FC Node_Name, and specific node descriptor data, if any, corresponding to the SRP initiator (see B.1.3.4.2). If an N_Port_ID is assigned to the SRP initiator, then the HBA shall also send a LOGO ELS to the fabric to release the N_Port_ID corresponding to the SRP initiator, provided that the N_Port_ID does not also correspond to another SRP initiator or FC management initiator. When the RMV field is set to one, the FC_PORT NAME_IDENTIFIER, NODE_NAME, and DATA-OUT BUFFER DESCRIPTOR fields are ignored by the HBA. The RMV field shall be set to zero if the bind for an SRP initiator is not to be removed.

The DATA-OUT BUFFMT field shall specify the format of the DATA-OUT BUFFER DESCRIPTOR (see B.1.2.11.2).

When the DATA-OUT BUFFER DESCRIPTOR field contains a valid data out buffer descriptor, then the data out buffer shall contain one or more data blocks containing the payload of an RNID accept that the HBA uses on behalf of the SRP initiator. Each data block contains the entire payload of an the RNID accept ELS to be sent in response to an RNID request ELS of a different type. If the Bind_SRP_Initiator request indicated that the FLOGI-assigned N_Port_ID is to be used on behalf of the SRP initiator, then the HBA FC_Port shall apply the specific node identification data to its FLOGI-assigned N_Port_ID; otherwise, the HBA FC_Port shall apply the specific node identification data to the N_Port_ID corresponding to the SRP initiator.

B.2.6.2 Bind_SRP_Initiator response

The Bind_SRP_Initiator response shall convey the response to the Bind_SRP_Initiator request. See table B.20 for the format of the Bind_SRP_Initiator response:

Table B.20 — Bind_SRP_Initiator response

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (A4h)							
1	REASON CODE							
2	reserved							
...								
7								
8	(MSB)							
...	TAG							
15								(LSB)
16	N_PORT_ID							
18								

If the Bind_SRP_Initiator was successfully performed and the ASGN bit was set to one, the N_PORT_ID field shall be set to the N_Port_ID that was assigned to the SRP initiator; if the ASGN bit is set to zero, the N_PORT_ID field shall be set to the FLOGI-assigned N_Port_ID. If the Bind_SRP_Initiator request was not successfully performed, the N_PORT_ID field shall be ignored.

The REASON CODE field shall indicate the completion status of the Bind_SRP_Initiator request. See table B.21 for a definition of the codes and their meanings.

Table B.21 — Bind_SRP_Initiator response reason codes

Reason Code	Description
00h	Request successfully completed
01h	One or more other Set_Port_Data or Bind_SRP_Initiator requests for this HBA FC_Port provided the same FC_Port Name, but the FC Node_Name provided is not the same as the FC Node_Name provided in the other requests. [a]
02h	N_Port_ID virtualization not supported; unable to store requested data
03h	Busy
04h	Insufficient resources to store the requested data fro the SRP initiator
05h	Unable to remove bind; An SRP login exists for the SRP initiator
06h	A Bind_SRP_Initiator has already been completed for this SRP_Initiator, and it has not been released.
07h	The HBA FC_Port is not attached to a fabric, unable to store requested data
08h	The ASGN bit was set to one, but the fabric was unable to provide an additional N_Port_ID
all other values	Reserved

[a] If more than one SRP initiator shares usage of the same N_Port_ID on an HBA, then the Bind_SRP_Initiator request for each SRP initiator shall contain the same FC_Port Name_Identifier and FC Node_Name. When FC management initiators share the N_Port_ID by providing the same FC_Port Name_Identifier in Set_Port_Data requests, their FC Node_Names shall also be identical.

B.2.7 Get_SRP_Initiator_Bindings

B.2.7.1 Get_SRP_Initiator_Bindings request

The Get_SRP_Initiator_Bindings request shall be used to request the SRP initiator binding information currently established for an HBA (i.e., the physical port number, N_Port_Name, N_Port_ID, Node_Name, and specific node identification data corresponding to each SRP initiator port that has been bound to an HBA virtual N_Port).

HBAs that support N_Port_ID virtualization shall support the Get_SRP_Initiator_Bindings request. HBAs that do not support N_Port_ID virtualization shall reject the Get_SRP_Initiator_Bindings request.

See table B.22 for the format of the Get_SRP_Initiator_Bindings request .

Table B.22 — Get_SRP_Initiator_Bindings request

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (08h)							
1	PHYSICAL PORT NUMBER							
2								
...	reserved							
6								
7				PPN	SII	DATA-IN BUFFMT		
8	(MSB)							
...	TAG							
15								(LSB)
16	(MSB)							
...	SRP INITIATOR PORT IDENTIFIER							
31								(LSB)
32	(MSB)							
...	DATA-IN BUFFER DESCRIPTOR							
47								(LSB)

If the PPN bit is set to one, the value of the PHYSICAL PORT NUMBER field shall be the number of a physical port on the HBA. If the PPN bit is set to zero, the value of the PHYSICAL PORT NUMBER field is not constrained by this standard.

If the SII bit is set to one, the value of the SRP INITIATOR PORT IDENTIFIER field shall be an SRP initiator ID. If the SII bit is set to zero, the value of the SRP INITIATOR PORT IDENTIFIER field is not constrained by this standard.

If the PPN bit is set to one the HBA shall return only established bindings for the physical port number that is specified in the PHYSICAL PORT NUMBER field. If the PPN bit is set to zero, the HBA shall return established bindings on the HBA for all physical ports.

If the SII bit is set to one the HBA shall return only established bindings for the SRP initiator ID that is specified in the SRP INITIATOR PORT IDENTIFIER field. If the SII bit is set to zero, the HBA shall return established bindings on the HBA for all SRP initiator ports.

If both the PPN bit and the SII bit are set to one the HBA shall return only an established binding of the physical port number that is specified in the PHYSICAL PORT NUMBER field to the SRP initiator ID that is specified in the SRP INITIATOR PORT IDENTIFIER field. If both the PPN bit and the SII bit are set to zero, the HBA shall return all established bindings on the HBA.

The DATA-IN BUFFMT field shall specify the format of the data-in buffer descriptor (see B.1.2.11.2).

The DATA-IN BUFFER DESCRIPTOR shall contain a buffer descriptor for a data buffer that the HBA shall use to store the SRP initiator bindings (see B.2.7.2).

B.2.7.2 Get_SRP_Initiator_Bindings response

The Get_SRP_Initiator_Bindings response shall be used to indicate completion of a request to transfer information regarding HBA SRP initiator bindings. See table B.23 for the format of the Get_SRP_Initiator_Bindings response

Table B.23 — Get_SRP_Initiator_Bindings response

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (A8h)							
1	REASON CODE							
2	reserved							
3								
4	(MSB)			AVAILABLE DATA				
...								
7								(LSB)
8	(MSB)			TAG				
...								
15								(LSB)

The REASON CODE field shall indicate the completion status of the Bind_SRP_Initiator request. See table B.24 for a definition of the codes and their meanings.

Table B.24 — Get_SRP_Initiator_Bindings response reason codes

Reason Code	Description
00h	Request successfully completed
01h	Reserved
02h	N_Port_ID virtualization not supported; unable to provide requested data
03h	Busy. The HBA shall not access the data-in buffer.
04h	The PPN bit is set to one, but the HBA is unable to find the specified physical port number.
04h-08h	Reserved
09h	The size of available data exceeds the size of the data-in buffer.
all other values	Reserved

The value of AVAILABLE DATA shall be the size in bytes of the entire list of SRP initiator bindings established by the HBA, regardless of the size of the data-in buffer.

The data-in buffer shall be used contain the SRP initiator binding list. The SRP initiator binding list shall contain all the SRP initiator bindings that are currently established for any SRP initiators for all HBA FC_Ports on the HBA unless it has been truncated to fit the data-in buffer. If the size of the entire SRP initiator binding list available from the HBA exceeds the size of the data-in buffer, the SRP initiator binding list shall be truncated after the last SRP initiator binding that completely fits in the data-in buffer, and the remaining bytes of the data-in buffer, if any, shall be set to zero. If the returned data is truncated, different SRP initiator bindings may be returned on successive requests. See table B.25 for the format of the SRP initiator binding.

Table B.25 — SRP initiator binding

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)	BINDING LENGTH						
1								(LSB)
2	PHYSICAL PORT NUMBER							
3	reserved							
4	(MSB)							
		SRP INITIATOR PORT IDENTIFIER						
19								(LSB)
20	(MSB)							
21		N_PORT_ID						
22								(LSB)
23	reserved							
24	(MSB)							
...		FC_PORT NAME_IDENTIFIER						
31								(LSB)
32	(MSB)							
...		FC NODE_NAME						
39								(LSB)
40	(MSB)							
...		SPECIFIC NODE IDENTIFICATION DATA						
n								(LSB)

The value in the BINDING LENGTH field shall be set to the length in bytes of the SRP initiator binding.

The PHYSICAL PORT NUMBER field shall be set to the physical port number of the HBA FC_Port to which the SRP initiator binding applies.

The SRP INITIATOR PORT IDENTIFIER field shall be set to the InfiniBand port identifier of the SRP initiator to which the SRP initiator binding applies.

The N_PORT_ID field shall be set to the N_Port_ID of the HBA virtual N_Port to which the SRP initiator binding applies.

The FC_PORT NAME_IDENTIFIER field shall be set to the N_Port_Name of the HBA virtual N_Port to which the SRP initiator binding applies.

The FC NODE_NAME field shall be set to the Node_Name of the HBA virtual N_Port to which the SRP initiator binding applies.

The SPECIFIC NODE IDENTIFICATION DATA field shall contain one or more data blocks containing the payload of an RNID accept that the HBA uses on behalf of the SRP initiator to which the SRP initiator binding applies. Each data block shall contain the entire payload of an the RNID accept ELS that is sent in response to an RNID request ELS of a different type.

B.2.8 Send_PassThru

B.2.8.1 Send_PassThru request

The Send_PassThru request allows the FC management initiator to send either an ELS request or an FC-CT request to a target FC_Port. It is also used to request the HBA to perform N_Port logout with a target FC_Port.

See table B.26 for the format of the Send_PassThru request.

Table B.26 — Send_PassThru request

Bit Byte	7	6	5	4	3	2	1	0
0	TYPE (05h)							
1	PHYSICAL PORT NUMBER							
2	(MSB)							
3	TARGET FC_PORT N_PORT_ID							
4								(LSB)
5	REQUEST TYPE							
6	reserved							

Table B.26 — Send_PassThru request

Bit Byte	7	6	5	4	3	2	1	0
7			DATA-OUT BUFFMT			DATA-IN BUFFMT		
8	(MSB)							
...				TAG				
15								(LSB)
16	(MSB)							
...				DATA-OUT BUFFER DESCRIPTOR				
31								(LSB)
32	(MSB)							
...				DATA-IN BUFFER DESCRIPTOR				
47								(LSB)

The PHYSICAL PORT NUMBER field shall be set to the physical port number of the HBA FC_Port from which the ELS request or FC-CT command is to be sent (see B.2.4.2).

The TARGET FC_PORT N_PORT_ID field shall indicate the N_Port_ID of the target FC_Port to which the ELS request or FC-CT command is to be sent.

The REQUEST TYPE field shall specify the type of request to be sent as shown in table B.27.

Table B.27 — Request Type Field

Value	Definition
00	fc els passthru
01	fc-ct command passthru
02	logout
02-FF	reserved

If the REQUEST TYPE field indicates FC ELS passthru or FC-CT command passthru, then the HBA shall determine if the N_Port_ID corresponding to the FC management initiator is currently logged in with the target FC_Port. If the N_Port_ID is logged in with the target FC_Port, the HBA shall send the ELS or FC-CT request to the destination N_Port_ID; if the N_Port_ID is not logged in with the target FC_Port, the HBA shall perform N_Port login with the target FC_Port prior to sending the ELS or FC-CT request.

If the REQUEST TYPE field indicates logout, then the HBA shall determine if there is an SRP login between an SRP initiator sharing usage of the N_Port_ID corresponding to the FC management initiator and the SRP target port identifier corresponding to the target FC_Port. If a login exists, then the HBA does not log out the N_Port_ID; if a login does not exist, the HBA shall log out the N_Port_ID corresponding to the FC management initiator with the target FC_Port.

The DATA-IN BUFFMT field shall specify the format of the DATA-IN BUFFER DESCRIPTOR. The DATA-OUT BUFFMT field shall specify the format of the DATA-OUT BUFFER DESCRIPTOR (see B.1.2.11.2).

The format of the DATA-IN BUFFER DESCRIPTOR and DATA-OUT BUFFER DESCRIPTOR fields shall be as specified in B.1.2.11.2. The data-out buffer shall contain the payload to be sent to the specified N_Port_ID. The data-in buffer shall used by the HBA to store the payload of the ELS response received from the target FC_Port to which the request was sent.

When the HBA receives a Send_PassThru request, the HBA shall send the data contained in the data-out buffer as the payload of an ELS or FC-CT command to the N_Port_ID of the target FC_Port. When the HBA receives the response, it shall place as much of the payload of the response in the data-in buffer as fits, and it shall send the Send_PassThru response payload to the FC management initiator.

B.2.7.2 Send_PassThru response

The Send_PassThru response shall convey the response to the Send_PassThru request. See table B.24 for the format of the Send_PassThru response:

Table B.24 — Send_PassThru response

Bit Byte	7	6	5	4	3	2	1	0
0	TYPE (A5h)							
1	REASON CODE							
2	reserved							
3								
4	(MSB)	AVAILBLE DATA						
...								
7								(LSB)
8	(MSB)	TAG						
...								
15								(LSB)

The REASON CODE field shall indicate the completion status of the Send_Passthru request See table B.25 for the definition of the reason codes.

The value of AVAILABLE DATA shall be the size in bytes of the entire ELS or FC-CT response payload, regardless of the size of the data-in buffer.

Table B.25 — Send_PassThru response reason codes

Reason Code	Description
00h	Request successfully completed
01h	Invalid ELS specified
02h	Target FC_Port is inaccessible
03h	Busy. The HBA shall not access the data-in buffer.
04h	N_Port logout was requested but not performed because an SRP login exists between an SRP initiator that shares the N_Port_D of the FC management initiator, and the SRP target port corresponding to the target FC_Port.
05h-08h	Reserved
09h	The size of available data exceeds the size of the data-in buffer.
all other values	Reserved

A reason code of Busy shall indicate that the request should be retried later.

A reason code of Invalid ELS shall be returned if the HBA does not support the sending of the ELS request. All HBAs shall support the LIRR and SCR ELSs. Support of all other ELSs depends on the model.

B.2.8 Get_Port_Statistics

B.2.8.1 Get_Port_Statistics request

The Get_Port_Statistics request allows the FC management initiator to obtain the end port statistics for the HBA FC_Port corresponding to the IB reliable connection. See table B.26 for the format of the Get_Port_Statistics request.

Table B.26 — Get_Port_Statistics request

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (06h)							
1	PHYSICAL PORT NUMBER							
2	reserved							
...								
7								
8	(MSB)							
...	TAG							
15								(LSB)

The PHYSICAL PORT NUMBER field shall be set to the physical port number of the HBA FC_Port to which the Set_Port_Data request is applied (see B.2.4.2).

B.2.8.2 Get_Port_Statistics response

The Get_Port_Statistics response shall contain the end port statistics for the HBA FC_Port corresponding to the IB reliable connection. See table B.27 for the format of the Get_Port_Statistics response.

Table B.27 — Get_Port_Statistics response

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (A6h)							
1	REASON CODE							
2	reserved							
...								
7								
8	(MSB)							
...	TAG							
15								(LSB)
16-23	reserved							
24-31	TXFRAMES							
32-39	RXFRAMES							
40-47	TXWORDS							
48-55	RXWORDS							
56-63	LIPCOUNT							
64-71	NOSCOUNT							
72-79	ERRORFRAMES							
80-87	DUMPEDFRAMES							
88-95	LINKFAILURECOUNT							
96-103	LossOfSynchCount							
104-111	LossOfSignalCount							
112-119	PrimitiveSeqProtocolErrCount							
120-127	InvalidTxWordCount							
128-135	InvalidCRCCount							
136-143	InputRequests							

Table B.27 — Get_Port_Statistics response

Bit Byte	7	6	5	4	3	2	1	0
144-151	OutputRequests							
152-159	ControlrRequests							
160-167	InputMegabytes							
168-175	OutputMegabytes							

The REASON CODE field shall indicate the completion status of the Get_Port_Statistics request See table B.28 for a definition of the codes and their meanings.

Table B.28 — Get_Port_Statistics response reason codes

Reason Code	Description
00h	Request successfully completed
01 - 02h	Reserved
03h	Busy. Bytes 16-175 shall be ignored.
all other values	Reserved

The end port statistics shall be contained in bytes 17-175 of the Get_Port_Statistics response. The end port statistics shall be as defined in subclause 6.5.2 of this standard. The values of all statistics are unsigned binary integers that wrap back to zero upon overflow.

A library may implement the HBA_Reset_Statistics function by storing the counter values during the HBA_Reset_Statistics function, and subtracting the current counter values from their stored values whenever counters are subsequently read.

B.2.9 Asynchronous Event (AEN) Reporting

B.2.9.1 AEN request

The AEN request shall be used by an HBA to report the receipt of an incoming ELS requests, and upon the occurrence of HBA_EVENT_PORT_OFFLINE and HBA_EVENT_PORT_ONLINE events, for the HBA FC_Port corresponding to the FC management connection.

Upon the occurrence of HBA_EVENT_PORT_OFFLINE and HBA_EVENT_PORT_ONLINE events, the AEN request shall be sent on all FC management connections in which the Establish_Connection request indicated that notification of asynchronous events is required (see B.2.2).

Upon receipt of an RSCN or RLIR ELS request to the FLOGI-assigned N_Port_ID, the HBA shall send an AEN request on each FC management connection in which the Establish_Connection request indicated that notification of asynchronous events is required. For HBAs that support N_Port_ID virtualization, the AEN request shall only be sent on the IB connections to the FC management initiators that have been assigned the N_Port_ID equal to the destination N_Port_ID of the incoming ELS request.

The HBA shall respond to incoming ELS requests as required by FC-FS.

See table B.29 for the format of the AEN request:

Table B.29 — AEN request

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (07h)							
	PHYSICAL PORT NUMBER							
1	(MSB)							
...				SOURCE N_PORT_ID				
3								(LSB)
4	EVENT TYPE							
5								
6				reserved				
7								
8	(MSB)							
...				TAG				
15								(LSB)
16	(MSB)			ELS PAYLOAD LENGTH				
								(LSB)
18	(MSB)							
...				ELS PAYLOAD				
n								(LSB)

The PHYSICAL PORT NUMBER field shall be set to the physical port number of the HBA FC_Port from which the AEN request was received (see B.2.4.2).

The EVENT TYPE field shall specify a code corresponding to the type of event that occurred. See table B.30 for a definition of the codes and their meanings

Table B.30 — Event Type Codes

Value	Meaning
00	els received
01	hba_event_port_online
02	hba_event_port_offline
03-FF	reserved

When the EVENT TYPE field specifies an ELS received event, the ELS PAYLOAD LENGTH field shall specify the length in bytes of the ELS Payload; if the EVENT TYPE field does not specify an ELS received event, the ELS PAYLOAD LENGTH field shall be ignored. The maximum value of the ELS PAYLOAD LENGTH field is 1 024.

The ELS PAYLOAD field contains the payload of the ELS that was received.

B.2.9.2 AEN response

The AEN response confirms the delivery of an AEN request. See table B.31 for the format of the AEN response:

Table B.31 — AEN response

Bit / Byte	7	6	5	4	3	2	1	0
0	TYPE (A7h)							
1	reserved							
...								
7								
8	(MSB) TAG							
...								
15								(LSB)

Since incoming RSCN ELSs contain the N_Port_IDs of the affected end ports instead of the FC Port Name_Identifiers, the library supporting the FC management initiator may need to convert each affected N_Port_ID into its corresponding SRP target port ID. To do this, the FC management initiator for the library may be used to access the nameserver in order to determine the affected FC_Port Name_Identifiers. Given this information, the library may construct the affected SRP target port identifiers as described in B.1.3.2.

Annex C

(Informative)

Target Mapping and Persistent Binding

C.1 Introduction to Target Mapping and Persistent Binding

C.1.1 The problem set

The basic problem to be resolved by Target Mapping and Persistent Binding functions is SCSI Identity Mapping: Current operating systems identify storage resources in different terms than the FCP-2 protocol (see FCP-2) does. One of the functions of an HBA and its driver is to translate between the OS and FCP-2 identities of storage resources. Specifying this translation is a basic part of managing the HBA (and the SAN behind it) since the translation determines which storage resources are presented to the OS by the HBA.

Subsidiary to this basic problem are several others, not all of which may be resolved by any particular HBA implementation:

a) Persistence: It may be desired that an OS identity reference the same FCP-2 storage resource despite reinitialization of the OS, HBA, and/or fabric.
b) Specificity: It may be desired to specify the OS identity assigned to a particular FCP-2 storage resource.
c) Legacy: It may be desired to support early implementations of identity mapping that are less functional or less widely accepted than the preferred methods today.
d) Autonomy: It may be desired that storage resources available via FCP-2 be made available to the OS without the need for administrative configuration.
e) Capacity: The number and/or type of storage resources accessible via FCP-2 may exceed the capacity of the OS or of an HBA driver to service.
f) Selectivity: It may be desired to control which FCP-2 resources are accessible by the OS or by a particular HBA.
g) Virtualization: More or less flexibility may be desired in making the structure of the storage system as viewed by the OS resemble that presented to the system by FCP-2 (this gets ahead of the presentation a little: Though the means of identifying storage resources differs between the OSs and FCP-2, both reflect the SCSI grouping of logical units within target devices. The question then relates to whether the groupings of logical units represented by the OS identifiers matches the groupings presented by FCP-2).

C.1.2 OS Identification of Storage Resources (SCSIID)

Most versions of Windows and Unix and their application programs identify storage resources via an abstraction of the classic SCSI Parallel Interface architecture (see SAM-3): A resource is identified 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, a character string.

The data structure HBA_SCSIID encapsulates the OS identification of a storage resource.

C.1.3 FCP-2 Identification of Storage Resources (FCPID)

FCP-2 is a standard SCSI-3 Service Delivery System (see SAM-3). As such, it provides identification for SCSI Initiator Ports and Target Ports, and for Logical Units behind Target Ports. Each Logical Unit is a distinct and indivisible storage resource.

In the Common HBA API specified by FC-MI, the Initiator Port is identified by the HBA handle via which FCP-2 management functions are exercised. In this standard, the Port Name of the HBA end port was added to resolve ambiguity in multiport adapters. The Target Port may be identified by any of its FC address identifier, its FC Port Name (WWPN), or its FC Node_Name (WWNN). The latter is intentionally ambiguous with respect to multi-port target devices, allowing the HBA and / or fabric to choose (hopefully to optimize the choice). The WWPN may be used when this flexibility is not desired.

Target Logical Units may be identified within a Target Port by their SCSI LUN, a 64-bit structured value defined in the SCSI Architecture Model specification (see SAM-3). The SCSI LUN is not the same as the OS LUN, though in some implementations one may be derived from the other.

Logical Units may also be identified independently of their containing Target Port by one of the several types of Vital Product Data (VPD) Page 83 identifiers defined by the SCSI Primary Commands specification (see SPC-3). This identity is intended to be independent of the means of accessing the Logical Unit (i.e., if a Logical Unit is accessible by several Service Delivery Systems, it should express the same VPD Page 83 identifiers on all of them). VPD Page 83 identifiers appear to be becoming the preferred means of identifying Logical Units within the SCSI standards community. Use of LUID alone is not practical in certain driver architectures and may not intuitively represent certain binding options (e.g., binding whole targets).

The data structures HBA_FCPID and HBA_LUID encapsulate the FCP-2 identification of a storage resource (SCSI target). HBA_FCPID represents the Target Port and SCSI LUN, while HBA_LUID represents a VPD Page 83 identifier. Either one is sufficient to identify a Target Logical Unit.

C.2 Target Mappings

Target Mappings resolve the Identity Mapping problem. A Target Mapping is a pairing of a SCSIID and an FCPID. It represents a relationship currently in effect such that SCSI operations requested by application programs with respect to the abstracted SCSI logical unit represented by the SCSIID act on the FCP logical unit identified by the FCPID. 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 Fibre Channel SAN resources. More than one mapping for any one SCSIID does not make sense, though more than one SCSIID may map to the same FCPID.

The HBA API specified by this standard provides no complete specification of how Target Mappings are established, though they may be affected by Persistent Bindings (see clause C.3).

C.3 Persistent Bindings

A Persistent Binding is a pairing of a SCSIID and FCPID that is retained through reinitialization of the OS, HBA, and / or fabric and establishes a Target Mapping subsequent to reinitialization. An HBA that supports Persistent Bindings therefore resolves the Persistence problem. An HBA that further has the capability to set and remove Persistent Bindings by functions defined in this standard also resolves the Specificity problem

This standard provides no complete specification of how Persistent Bindings are established, though they may be affected by the FCP Information Functions it specifies (see 7.4).

C.4 Persistent Binding Capabilities

C.4.1 Overview

A Persistent Binding Capability represents the ability of an HBA to provide a specific feature 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.

Management of Persistent Binding Capabilities ameliorates the Legacy problem: Capabilities formalize optional support for features that are desirable but not yet implemented and features that are already implemented but not likely to be widely adopted.

C.4.2 Persistent Binding Capability: HBA_CAN_BIND_TO_D_ID

The Persistent Binding capability HBA_CAN_BIND_TO_D_ID represents the ability of an HBA to accept a Persistent Binding that identifies the Fibre Channel target port by its Fibre Channel address identifier. In larger and more dynamic SANs, it is not necessarily persistent over the time frames that may be of interest in establishing Persistent Bindings. It also may not be unique across disjoint fabrics, which may raise identity ambiguities in multi-fabric SANs. Address identifier should be used only where it is the only alternative, or its persistence and uniqueness are known by the local administration to be sufficient. Its capability should be considered primarily an amelioration of the Legacy problem

C.4.3 Persistent Binding Capability: HBA_CAN_BIND_TO_WWPN

The Persistent Binding capability HBA_CAN_BIND_TO_WWPN represents the ability of an HBA to accept a Persistent Binding that identifies the Fibre Channel target port by its WWPN. The WWPN is a Fibre Channel Name_Identifier for a specific end port. Its persistence and uniqueness are sufficient for all expected uses.

C.4.4 Persistent Binding Capability: HBA_CAN_BIND_TO_WWNN

The Persistent Binding capability HBA_CAN_BIND_TO_WWNN represents the ability of an HBA to accept a Persistent Binding that identifies a Fibre Channel target device (not a target port) by its World Wide Node Name (WWNN). The WWNN is a Fibre Channel Name_Identifier presumed to apply to a single FCP-2 device. Its ambiguity with respect to multi-port devices is intentional, being left for the HBA and / or fabric to resolve, so though not necessarily unique, it is sufficiently specific. Its persistence is sufficient for all expected uses.

C.4.5 Persistent Binding Capability: HBA_CAN_BIND_TO_LUID

The Persistent Binding capability HBA_CAN_BIND_TO_LUID represents the ability of an HBA to accept a Persistent Binding that identifies the Fibre Channel target logical unit by the value of one of its device-associated Identification Descriptors (LUID). The LUID may be a Name_Identifier, an EUI-64, a T10 vendor identification, or a vendor specific value. Its persistence and uniqueness are sufficient for all expected uses if it is a Name_Identifier, an EUI-64, or a T10 vendor identification. A vendor specific LUID has no assurance of uniqueness or persistence. One should be used only if it is the only alternative, or its persistence and uniqueness are known by the local administration to be sufficient. Vendor Specific LUIDs and T10 vendor identification LUIDs are supported only to resolve an aspect of the Legacy problem.

C.4.6 Persistent Binding Capability: HBA_CAN_BIND_ANY_LUNS

The Persistent Binding capability HBA_CAN_BIND_ANY_LUNS represents the ability of an HBA to accept Persistent Binding settings that independently specify both the OS and Fibre Channel target LUNs. This may be used when FCP LUN values exceed the maximum value permitted for an OS LUN. More powerfully, these settings allow storage resources to be grouped differently by SCSIID than they are grouped by FCPID. This means that logical units provided on a single FCP target may be represented to the OS with different Target Numbers, and logical units provided on different FCP targets may be represented to the OS with the same Target Number. This capability is intended to assist in resolving the Virtualization problem.

An HBA that does not express the HBA_CAN_BIND_ANY_LUNS capability may require that all Persistent Binding settings preserve the groupings of logical units into devices (i.e., for any pair of Persistent Binding settings, the SCSIIDs may designate the same target device if and only if the FCPIDs designate the same end port).

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.

C.4.7 Persistent Binding Capability: HBA_CAN_BIND_TARGETS

The Persistent Binding capability HBA_CAN_BIND_TARGETS represents the ability of an HBA to accept a Persistent Binding setting with type including HBA_BIND_TARGETS. This reflects support for device-level mappings (i.e., the HBA is able to automatically generate Target Mappings from OS LUNs on the OS controller and target indicated by the SCSIID to all logical units on the FCP target port indicated by FCPID). This capability is intended to assist in resolving the Virtualization and Autonomy problems.

C.4.8 Persistent Binding Capability: HBA_CAN_ BIND_AUTOMAP

The Persistent Binding capability HBA_CAN_BIND_AUTOMAP indicates that an HBA is able to attempt to automatically generate Target Mappings and Persistent Bindings for all discovered storage resources. HBAs with this capability, especially in combination with small SANs and self-identifying file systems, may provide service without administrative intervention, resolving the Autonomy problem.

If this capability is not indicated (or disabled), the HBA is only able to establish Target Mappings based on Persistent Bindings that have been explicitly set (sometimes described as LUN Masking). This may be useful in order to limit the impact on hosts of very large SANs. Disabling this capability provides resolution for both the Capacity and Selectivity problems.

C.4.9 Persistent Binding Capability: HBA_CAN_BIND_CONFIGURED

The Persistent Binding capability HBA_CAN_BIND_CONFIGURED represents the ability of an HBA to accept the Persistent Binding configuration functions HBA_SetPersistentBindingV2, HBA_RemovePersistentBinding, and HBA_RemoveAllPersistentBindings.

An HBA that does not express this capability may provide only some form of automatically generated Persistent Bindings.

Annex D
(informative)

Function Coding Examples

D.1 Function HBA_GetVersion

Following is an example usage of HBA_GetVersion in an application program:

```
HBA_UINT32 version;

version = HBA_GetVersion();

printf(“Running version %d of the HBA API library.”, version);
```

D.2 Function HBA_LoadLibrary

Following is an example usage of HBA_LoadLibrary in an application program:

```
HBA_STATUS status;

status = HBA_LoadLibrary();

printf(“Successfully loaded HBA library.\n”);
```

D.3 Function HBA_FreeLibrary

Following is an example usage of HBA_FreeLibrary in an application program:

```
HBA_STATUS status;

status = HBA_FreeLibrary();

printf(“Successfully freed  HBA library.\n”);
```

D.4 Function HBA_RegisterLibrary

Following is an example implementation of HBA_RegisterLibrary in an HBA specific library:

```
/* Initialize pointers to our versions of the common HBA API */

HBA_ENTRYPOINTS HBAInfo;

memset(&HBAInfo, 0, sizeof(HBA_INFO));
HBAInfo.GetVersionHandler = MYGetVersion;
HBAInfo.GetNumberOfAdaptersHandler = MYGetNumberOfAdapters;
HBAInfo.GetAdapterNameHandler = MYGetAdapterName;
HBAInfo.OpenAdapterHandler = MYOpenAdapter;
HBAInfo.CloseAdapterHandler = MYCloseAdapter;
HBAInfo.GetAdapterAttributesHandler = MYGetAdapterAttributes;
HBAInfo.GetAdapterPortAttributesHandler = MYGetAdapterPortAttributes;
HBAInfo.GetPortStatisticsHandler = MYGetPortStatistics;
HBAInfo.GetDiscoveredPortAttributesHandler = MYGetDiscoveredPortAttributes;
HBAInfo.GetPortAttributesByWWNHandler = MYGetPortAttributesByWWN;
HBAInfo.RefreshInformationHandler = MYGetRefreshInformation;
HBAInfo.InitiateLIPHandler = NULL; /* Not supported */
HBAInfo.GetFcpTargetMappingHandler = NULL;
HBAInfo.GetFcpPersistentBindingHandler = NULL;
HBAInfo.GetEventBufferHandler = NULL;
HBAInfo.SetRNIDMgmtAddressHandler= NULL;
HBAInfo.GetRNIDMgmtAddressHandler = NULL;
HBAInfo.SendRNIDHandler= NULL;
HBAInfo.ScsiInquiryHandler= NULL;
HBAInfo.ReportLUNsHandler= NULL;
HBAInfo.ReadCapacityHandler= NULL;

return HBA_STATUS_OK;
```

D.5 Function HBA_GetNumberOfAdapters

Following is an example usage of HBA_GetNumberOfAdapters and HBA_GetAdapterName in an application program:

```
HBA_UINT32 version;
int i;
HBA_STATUS status;
char adaptername[256];

      number_of_adapters = HBA_GetNumberOfAdapters();

      for (i = 0; i < number_of_adapters; i++) {

            status = HBA_GetAdapterName(i, &adaptername);
            if (status == HBA_STATUS_OK) {
                  printf("Adapter %d is named %s\r\n", i, adaptername);
            }
      )
```

D.6 Function HBA_RefreshInformation

Following is an example usage of HBA_RefreshInformation in an application program:

```
HBA_RefreshInformation(adapterhandle);
status = HBA_GetPortStatistics(adapterhandle, portindex, &portstats);
```

D.7 Function HBA_GetAdapterName

Following is an example usage of HBA_GetNumberOfAdapters and HBA_GetAdapterName in an application program:

```
HBA_UINT32 version;
int i;
HBA_STATUS status;
char adaptername[256];

      number_of_adapters = HBA_GetNumberOfAdapters();

      for (i = 0; i < number_of_adapters; i++) {

            status = HBA_GetAdapterName(i, &adaptername);
```

```
                if (status == HBA_STATUS_OK) {
                      printf(“Adapter %d is named %s\r\n”, i, adaptername);
                }
          )
```

D.8 Function HBA_OpenAdapter

Following is an example usage of HBA_GetNumberOfAdapters, HBA_GetAdapterName, HBA_OpenAdapter, and HBA_CloseAdapter in an application program:

```
int i;
HBA_STATUS status;
HBA_HANDLE adapterhandle;
char adaptername[256];

      number_of_adapters = HBA_GetNumberOfAdapters();

      for (i = 0; i < number_of_adapters; i++) {

            status = HBA_GetAdapterName(i, &adaptername);
            if (status == HBA_STATUS_OK) {
                  adapterhandle = HBA_OpenAdapter(adaptername);
                  if (adapterhandle != NULL) {
                        printf(“Successfully opened %s\r\n”,
                              adaptername);
                        HBA_CloseAdapter(adapterhandle);
                  }
            }
      )
```

D.9 Function HBA_CloseAdapter

Following is an example usage of HBA_OpenAdapter and HBA_CloseAdapter in an application program:

```
adapterhandle = HBA_OpenAdapter(adaptername);
if (adapterhandle != NULL) {
      printf(“Successfully opened %s\r\n”, adaptername);
      HBA_CloseAdapter(adapterhandle);
}
```

D.10 Function HBA_GetAdapterAttributes

Following is an example usage of HBA_GetAdapterAttributes in an application program:

```
HBA_STATUS status;
HBA_ADAPTERATTRIBUTES adapterattributes;

status = HBA_GetAdapterAttributes(adapterhandle, &adapterattributes);

printf(“Manufacturer: %s\r\n”, adapterattributes.Manufacturer);
printf(“Serial Number: %s\r\n”, adapterattributes.SerialNumber);
```

D.11 Function HBA_GetAdapterPortAttributes

Following is an example usage of HBA_GetAdapterAttributes and HBA_GetAdapterPortAttributes in an application program:

```
HBA_STATUS status;
HBA_ADAPTERATTRIBUTES adapterattributes;
HBA_PORTATTRIBUTES portattributes;

status = HBA_GetAdapterAttributes(adapterhandle, &adapterattributes);

for (i = 0; i < adapterattributes.NumberOfPorts; i++) {

       status = HBA_GetAdapterPortAttributes(adapterhandle, i,
                                           &portattributes);

       if (status == HBA_STATUS_OK) {
              printf(“Port %d has an FC-FS address identifier of %d”, i,
                     portattributes.PortFcId);
       }
}
```

D.12 Function HBA_GetDiscoveredPortAttributes

Following is an example usage of HBA_GetDiscoveredPortAttributes in an application program:

```
HBA_STATUS status;
HBA_PORTATTRIBUTES portattributes;

/* Get the attributes of first discovered FC_Port on first HBA port */
status = HBA_GetDiscoveredPortAttributes(handle, 1, 1, &portattributes);
if (status == HBA_STATUS_OK) {
       printf(“FC_Port 1 on HBA Port 1 ”,
       printf(“has an FC-FS address identifier of %d”,
              portattributes.PortFcId);
}
```

D.13 Function HBA_GetPortStatistics

Following is an example usage of HBA_GetPortStatistics in an application program:

```
HBA_STATUS status;
HBA_PORTSTATISTICS portstats;

status = HBA_GetPortStatistics(adapterhandle, portindex, &portstats);

if (status == HBA_STATUS_OK) {
       printf(“Port %d has sent %d frames.”, portindex, portstats.TxFrames);
}
```

Annex E
(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 20 The SNIA is the Storage Networking Industry Association. Information about the availability of its publications may be obtained from the SNIA by writing to 2570 West El Camino Real Suite 30, Mountain View, CA 94040-1313 USA, by telephone to (USA) 650.949.6750, or on the World Wide Web at http://www.snia.org/

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