Fibre Channel HBA API

SNIA FC Working Group
HBA API Subgroup
Chair: Benjamin F. Kuo,
Troika Networks, Inc.

1.0
9/11/2000

Version	Date	Change	Author
1.0	9/11/2000	1.0 Final	Benjamin Kuo

1 Contributors

Companies instrumental in helping to define this API include:

Agilent Technologies
Emulex Corporation
JNI Corporation
QLogic Corporation
Troika Networks, Inc.

I'd like to thank the following individuals for their contributions to shaping this paper:

Duane Baldwin, Tivoli Systems
David Chambliss, IBM Research Division
Don Deel, Prisa Networks
Mike Dutch, Troika Networks Inc.
Jeff Goldner, Microsoft
Ying Ping Lok, Qlogic Corporation
Tuan Lam, Qlogic Corporation
Alan Land, JNI Corporation
Chris Mercier, Intersan
Arun Mittal, Qlogic Corporation
Bob Nixon, Emulex Corporation
Ie Wei Njoo, Agilent Technologies
Darrell Smith, JNI Corporation
Predrag Spasic, Hewlett Packard
Bill Terrell, Troika Networks Inc.
Paul Timmins, StorageNetworks
Dan Willie, Emulex Corporation
Steve Wilson, Brocade

2 Introduction

A Fibre Channel Host Bus Adapter is a piece of hardware, typically on a host system and sometimes embedded on a RAID controller or other storage device, which interfaces a system to the Fibre Channel Network. An adapter can support multiple FC-4 types, including FCP-SCSI, IPFC, and FC-VI.

The problem exists today where, in order to operate, upper level software applications require information which is not available from host bus adapters in a consistent manner across operating systems, vendors, and platforms, and in some cases not at all.

Current workarounds to this problem have been HBA vendor specific; that is, when a software application needs access to certain Fibre Channel parameters (i.e. WWN or attached LUNs), special drivers or OS specific calls have had to be utilized to get to this information. This results in long qualification times, difficult integration across platforms, and inconsistency between HBA vendors. This makes implementation of SAN applications almost impossible for upper level software applications in an efficient and consistent manner.

Current existing CIM level models, although useful for management, are at a much higher level than this proposal which is targeted at providing lower level operational information to applications. This information is a superset of the current CIM model and should be migrated over the long term into the CIM model.

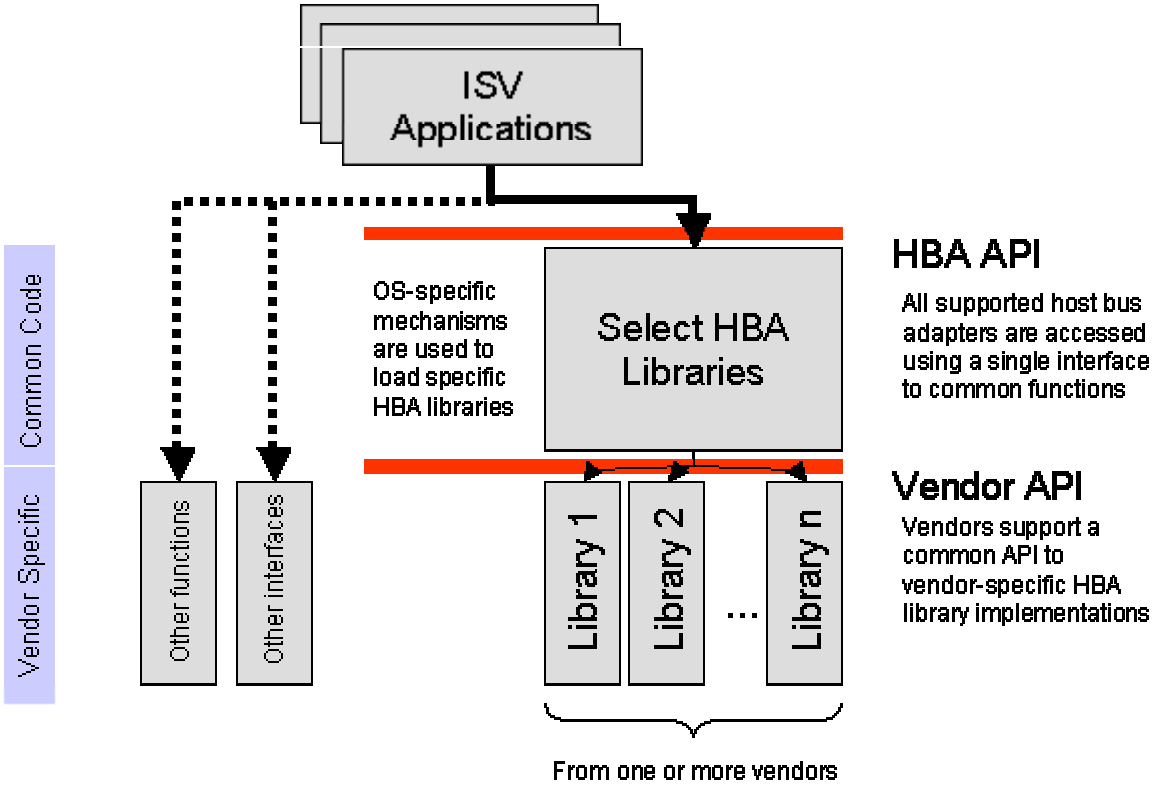
This is a proposal to adopt a low level, consistent HBA standard interface for accessing information in a Fibre Channel Storage Area Network, which would be implemented across vendors as a standard 'C' API supported by vendor specific library instances.

Some specific operations which an HBA API needs to support are:

- Ability to query local HBA properties and port information
- Ability to correlate a WWN of a device to the local SCSI address
- Ability to discover the WWNs of discovered end nodes
- Ability to query properties of discovered end nodes
- Ability to retrieve LUN mappings of an HBA

3 HBA API Model

The HBA API is a single C-style library interface. At the upper level, a common HBA library provides the ability to handle multiple vendor implementations of the HBA API through dynamic loading of libraries. At the intermediate level, an identical but vendor-provided library provides vendor-specific implementations of the HBA API. Calls at the common HBA library API are identical in form to the Vendor Implemented HBA Libraries at the API interface, but implementation of how to provide information to the API is left to each vendor.



4 HBA Information Definitions

All Fibre Channel HBAs have certain information in common, regardless of manufacturer. Upper level applications need access to this information, which needs to be presented in a consistent manner.

4.1 HBA Information Definitions

Overview:

Adapter Attributes
- Manufacturer
- Serial Number
- Model
- Model Description
- Node WWN
- Node Symbolic Name
- Hardware Version
- Driver Version
- Option ROM Version
- Firmware Version
- Vendor Specific ID
- Number Of Ports
- Driver Name

Port Attributes
- Node WWN
- Port WWN
- Port Symbolic Name
- Port FcId
- Port Type (Point-to-Point, N-port, L-port, NL-port)
- Port State
- Port Supported Class of Service
- Port Supported Fc4Types
- Port Active Fc4Types
- Port Supported Speed
- Port Speed
- Port Max Frame Size
- OS Device Name
- Number of Discovered Ports
- Fabric Name

Port Statistics
- Seconds Since Statistics Reset
- TxFrames
- RxFrames
- TxWords
- RxWords
- LIPCount
- NOSCount
- ErrorFrames
- DumpedFrames
- LinkFailureCount
- LossOfSyncCount
- LossOfSignalCount
- PrimitiveSeqProtocolErrCount
- InvalidTxWordCount
- InvalidCRCCount

Port FCP Attributes
- Node WWN
- Port WWN
- FcId
- FcpLun
- OSDeviceName
- ScsiBusNumber
- ScsiTargetNumber
- ScsiOSLun
- FCP Binding Type

FC-3 Management Attributes
- WWN
- unittype
- PortId
- NumberOfAttachedNodes
- IPVersion
- UDPPort
- IPAddress
- TopologyDiscoveryFlags
- TimeStamp
- EventCode

4.1.1 Adapter Attributes

4.1.1.1 Manufacturer

The Manufacturer is an ASCII string indicating the manufacturer of an HBA. This attribute is a null-terminated ASCII string with length from 1 to 64 bytes (including the null).

Example:
Emulex

4.1.1.2 Serial Number

The Serial Number is an ASCII string indicating the serial number of an HBA. This attribute is a null-terminated ASCII string with length from 1 to 64 bytes (including the null).

Example:
A012345

4.1.1.3 Model

The Model is a textual description of the model. This attribute is a null-terminated ASCII string with length from 1 to 256 bytes (including the null).

Example:
QLA2200

4.1.1.4 Model Description

The Model Description is a longer textual description of the model. This attribute is a null-terminated ASCII string with length from 1 to 256 bytes (including the null).

Example:
Agilent Tachlite Fibre Channel Adapter

4.1.1.5 Node WWN

The Node WWN is an 8 byte value indicating the Node WWN of this HBA.

4.1.1.6 Node Symbolic Name

The Node Symbolic Name is a null terminated ASCII string corresponding to the Fibre Channel Node Symbolic Name. This attribute is a null-terminated ASCII string with length from 1 to 256 bytes (including the null).

4.1.1.7 Hardware Version

The Hardware Version is a vendor specific null terminated ASCII string indicating the hardware revision level of the HBA. This string may differ from manufacturer to manufacturer, and cannot be parsed.

4.1.1.8 Driver Version

The Driver Version is a vendor specific string indicating the driver version controlling this HBA. This string may differ from manufacturer to manufacturer, and cannot be parsed. This attribute is a null-terminated ASCII string with length from 1 to 256 bytes (including the null).

4.1.1.9 Option ROM Version

The Option ROM Version is a vendor specific ASCII string indicating the option ROM or BIOS version of the HBA, if any. This string may differ from manufacturer to manufacturer, and cannot be parsed. This attribute is a null-terminated ASCII string with length from 1 to 256 bytes (including the null).

4.1.1.10 Firmware Version

The Firmware Version is a vendor specific ASCII string indicating the firmware version of the HBA, if any. This string may differ from manufacturer to manufacturer, and cannot be parsed. This attribute is a null-terminated ASCII string with length from 1 to 256 bytes (including the null).

4.1.1.11 Vendor Specific ID

The Vendor Specific ID is a vendor specific field.

4.1.1.12 Number of Ports

The Number of Ports is the number of ports on this adapter.

4.1.1.13 Driver Name

The Driver Name is an ASCII string denoting the file name for the the driver binary file. In the case of some operating systems that implement a generic driver name (such as Driver.o in Unixware) an absolute path could be included in the driver name. This attribute is a null-terminated ASCII string with length from 1 to 256 bytes (including the null).

Example:

For NT 4.0 or Win2000 environment, it is the SCSI miniport driver name for the HBA. For example, AFCW2K.SYS is the name of the binary file for the SCSI miniport for the Agilent FC HBA.

For UnixWare that uses generic driver name Driver.o, the full/absolute path could be used.
Example: /etc/conf/pack.d/Agilent/Driver.o

4.1.2 Port Attributes

4.1.2.1 Node WWN

The Node WWN is an 8-byte value corresponding to the Fibre Channel Node WWN associated with this port.

4.1.2.2 Port WWN

The Port WWN is an 8-byte value corresponding to the Fibre Channel Port WWN of this port.

4.1.2.3 Port Symbolic Name

The Port Symbolic name is a null terminated ASCII string which corresponds to the Fibre Channel Port Symbolic Name. This attribute is a null-terminated ASCII string with length from 1 to 256 bytes (including the null).

4.1.2.4 Port FcId

The Port FcId is a 3-byte identifier which corresponds to the current Fibre Channel address of the port.

4.1.2.5 Port Type

The Port Type is an enumerated type which corresponds to the current operational mode of the port.

4.1.2.6 Port State

The Port State is an enumerated type which corresponds to the current state of the port.

4.1.2.7 Port Supported Class of Service

The supported Class of Service is a bitmask of the supported class of service of this port.

4.1.2.8 Port Supported Fc4Types

The PortSupportedFc4Types is a bitmask of the supported FC4 types of this port.

4.1.2.9 Port Active Fc4Types

The PortActiveFc4Types is a bitmask of the current FC4 types on this port.

4.1.2.10 Port Supported Speed

The PortSupportedSpeed is a bitmask of the supported speeds of this interface.

4.1.2.11 Port Speed

The Port Speed is a bitmask of the current speed of the interface.

4.1.2.12 Port Max Frame Size

The Port Max Frame Size is the maximum frame size of this port.

4.1.2.13 OS Device Name

The OS Device Name is a null terminated ASCII string describing the device name that this port is visible from on the operating system, if known. This attribute is a null-terminated ASCII string with length from 1 to 256 bytes (including the null).

4.1.2.14 Number Of Discovered Ports

For a local port, the number of discovered ports that this port can communicate with. For discovered ports this value is not used.

4.1.2.15 Fabric Name

The WWN for the fabric to which the port is attached, if known.

4.1.3 Port Statistics

4.1.3.1 SecondsSinceLastReset

SecondsSinceLastReset is number of seconds since the statistics were last reset.

4.1.3.2 TxFrames

TxFrames is number of total Transmitted Fibre Channel frames across all protocols and classes.

4.1.3.3 RxFrames

RxFrames is the number of total Received Fibre Channel frames across all protocols and classes.

4.1.3.4 TxWords

TxWords is the number of total Transmitted Fibre Channel words across all protocols and classes.

4.1.3.5 RxWords

RxWords is the number of total Received Fibre Channel words across all protocols and classes.

4.1.3.6 LIPCount

LIPCount is the number of LIP events that have occurred on a arbitrated loop.

4.1.3.7 NOSCount

NOSCount is the number of NOS events that have occurred on the switched fabric.

4.1.3.8 ErrorFrames

ErrorFrames is the number of frames which have been received in error.

4.1.3.9 DumpedFrames

DumpedFrames is the number of frames which were dumped due to a lack of host buffers available.

4.1.3.10 LinkFailureCount

LinkFailureCount is the number of times a link error has occurred.

4.1.3.11 LossOfSyncCount

LossOfSyncCount is the number of times loss of synch has occurred.

4.1.3.12 LossOfSignalCount

LossOfSignalCount is the number of times loss of signal has occurred.

4.1.3.13 PrimitiveSeqProtocolErrCount

Primitive Sequence Protocol Error Count is the number of primitize sequence protocol errors.

4.1.3.14 InvalidTxWordCount

InvalidTxWordCount is the number of invalid transmitted words.

4.1.3.15 Invalid CRC Count

InvalidCRCCount is the number of frames received with invalid CRC.

4.1.4 Port FCP Attributes

4.1.4.1 Node WWN

The Node WWN of the end device.

4.1.4.2 Port WWN

The Port WWN of the end device.

4.1.4.3 FcId

The FcId of the device in a binding context.

4.1.4.4 FcpLun

The full FC Lun value of this device.

4.1.4.5 OSDeviceName

The device name of this LUN as presented at the operating system, if known. This attribute is a null-terminated ASCII string with length from 1 to 256 bytes (including the null).

4.1.4.6 ScsiBusNumber

The SCSI bus number of this LUN as presented at the operating system, if known.

4.1.4.7 ScsiTargetNumber

The SCSI Target Number of this LUN as presented to the operating system.

4.1.4.8 ScsiOSLun

The operating system LUN this FC LUN is presented as to the operating system.

4.1.4.9 FCP Binding Type

The type of binding - to the WWN or to the FcId of this LUN.

4.1.5 FC-3 Management Attributes

4.1.5.1 WWN

Corresponds with RNID WWN field, indicating the world wide name of this node.

4.1.5.2 unittype

Corresponds with RNID unit type field, describing the type of equipment this HBA represents.

4.1.5.3 PortId

Corresponds with RNID Port ID field.

4.1.5.4 NumberOfAttachedNodes

Corresponds with RNID Number of Attached Nodes field.

4.1.5.5 IPVersion

Corresponds with the RNID IP Version field, indicating whether the following IP address is a IPv4 or IPv6 address.

4.1.5.6 UDPPort

Corresponds with the RNID UDP Port field, indicating the management UDP port.

4.1.5.7 IPAddress

Corresponds to the RNID IP address field, indicating the management IP address.

4.1.5.8 TopologyDiscoveryFlags

Corresponds to the RNID Topology Discovery Flags field.

4.1.5.9 TimeStamp

Time Stamp contains a time stamp of events received from the adapter. This value is the number of seconds since the last driver reboot when the event occurred.

4.1.5.10 EventCode

Event Code contains an event code describing the event that occurred.

4.2 Data Structures Definition

4.2.1 Handle to Device

```
typedef HBA_UINT32 HBA_HANDLE;
```

4.2.2 Status Return Values

```
typedef HBA_UINT32 HBA_STATUS;

#define HBA_STATUS_OK                       0
#define HBA_STATUS_ERROR                    1   /* Error */
#define HBA_STATUS_ERROR_NOT_SUPPORTED      2   /* Function not supported.*/
#define HBA_STATUS_ERROR_INVALID_HANDLE     3   /* invalid handle */
#define HBA_STATUS_ERROR_ARG                4   /* Bad argument */
#define HBA_STATUS_ERROR_ILLEGAL_WWN        5   /* WWN not recognized */
#define HBA_STATUS_ERROR_ILLEGAL_INDEX      6   /* Index not recognized */
#define HBA_STATUS_ERROR_MORE_DATA          7   /* Larger buffer required */
```

4.2.3 Port Operational Modes Values

```
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 /* Fabric  */
#define HBA_PORTTYPE_NLPORT             6 /* Public Loop */
#define HBA_PORTTYPE_FLPORT             7
#define HBA_PORTTYPE_FPORT              8 /* Fabric Port */
#define HBA_PORTTYPE_EPORT              9 /* Fabric expansion port */
#define HBA_PORTTYPE_GPORT             10 /* Generic Fabric Port */
#define HBA_PORTTYPE_LPORT             20 /* Private Loop */
#define HBA_PORTTYPE_PTP               21 /* Point to Point */

typedef HBA_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 */

typedef HBA_UINT32 HBA_PORTSPEED;
#define HBA_PORTSPEED_1GBIT             1 /* 1 GBit/sec */
#define HBA_PORTSPEED_2GBIT             2 /* 2 GBit/sec */
#define HBA_PORTSPEED_10GBIT            4 /* 10 GBit/sec */
```

4.2.4 Class of Service Values

typedef HBA_UINT32 HBA_COS;
See GS-2 Spec.

4.2.5 Fc4Types Values

```
typedef struct HBA_fc4types {
        HBA_UINT8 bits[32]; /* 32 bytes of FC-4 per GS-2 */
} HBA_FC4TYPES, *PHBA_FC4TYPES;
```
See GS-2 Spec.

4.2.6 Basic Types

```
typedef struct HBA_wwn {
        HBA_UINT8 wwn[8];
} HBA_WWN, *PHBA_WWN;

typedef struct HBA_ipaddress {
        int     ipversion;              // see enumerations in RNID
        union
        {
                unsigned char ipv4address[4];
                unsigned char ipv6address[16];
        } ipaddress;
} HBA_IPADDRESS, *PHBA_IPADDRESS;
```

4.2.7 Adapter Attributes

```
typedef struct HBA_AdapterAttributes {
```

```
        char                    Manufacturer[64];          /*Emulex */
        char                    SerialNumber[64];          /* A12345 */
        char                    Model[256];                * QLA2200 */
        char                    ModelDescription[256];     /* Agilent TachLite */
        HBA_WWN                 NodeWWN;
        char                    NodeSymbolicName[256];  /* From GS-2 */
        char                    HardwareVersion[256];      /* Vendor use */
        char                    DriverVersion[256];        /* Vendor use */
        char                    OptionROMVersion[256];  /* Vendor use  - i.e. hardware boot ROM*/
        char                    FirmwareVersion[256];      /* Vendor use */
        HBA_UINT32              VendorSpecificID;          /* Vendor specific */
        HBA_UINT32              NumberOfPorts;
        char                    DriverName[256];           /* Binary path and/or name of driver file. */
} HBA_ADAPTERATTRIBUTES, *PHBA_ADAPTERATTRIBUTES;
```

4.2.8 Port Attributes

```
typedef struct HBA_PortAttributes {
        HBA_WWN                 NodeWWN;
        HBA_WWN                 PortWWN;
        HBA_UINT32              PortFcId;
        HBA_PORTTYPE            PortType;                  /*PTP, Fabric, etc. */
        HBA_PORTSTATE           PortState;
        HBA_COS                 PortSupportedClassofService;
        HBA_FC4TYPES            PortSupportedFc4Types;
        HBA_FC4TYPES            PortActiveFc4Types;
        char                    PortSymbolicName[256];
        char                    OSDeviceName[256];         /* \device\ScsiPort3  */
        HBA_PORTSPEED           PortSupportedSpeed;
        HBA_PORTSPEED           PortSpeed;
        HBA_UINT32              PortMaxFrameSize;
        HBA_WWN                 FabricName;
        HBA_UINT32              NumberofDiscoveredPorts;
} HBA_PORTATTRIBUTES, *PHBA_PORTATTRIBUTES;
```

4.2.9 Port Statistics

```
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;
```

4.2.10 Port FCP Attributes

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

typedef struct HBA_ScsiId {
```

```
        char                    OSDeviceName[256];      /* \device\ScsiPort3  */
        HBA_UINT32              ScsiBusNumber;          /* Bus on the HBA */
        HBA_UINT32              ScsiTargetNumber;       /* SCSI Target ID to OS */
        HBA_UINT32              ScsiOSLun;
} HBA_SCSIID, *PHBA_SCSIID;

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

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

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

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

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

4.2.11 FC-3 Management Atrributes

```
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;

#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

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

```
typedef struct HBA_RSCN_EventInfo {
        HBA_UINT32 PortFcId;     /* Port which this event occurred */
        HBA_UINT32 NPortPage;   /* Reference FC-FS for
                                                RSCN ELS "Affected N-Port 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;
```

4.2.12 HBA Library Function Table

```
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,
PHBA_ADAPTERATTRIBUTES);
typedef HBA_STATUS      ( * HBAGetAdapterPortAttributesFunc)(HBA_HANDLE, HBA_UINT32,
PHBA_PORTATTRIBUTES);
typedef HBA_STATUS      ( * HBAGetPortStatisticsFunc)(HBA_HANDLE, HBA_UINT32,
PHBA_PORTSTATISTICS);
typedef HBA_STATUS      ( * HBAGetDiscoveredPortAttributesFunc)(HBA_HANDLE, HBA_UINT32,
HBA_UINT32, PHBA_PORTATTRIBUTES);
typedef HBA_STATUS      ( * HBAGetPortAttributesByWWNFunc)(HBA_HANDLE, HBA_WWN,
PHBA_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,
PHBA_FCPTARGETMAPPING );
typedef HBA_STATUS      ( * HBAGetFcpPersistentBindingFunc) (HBA_HANDLE, PHBA_FCPBINDING );
typedef HBA_STATUS      (* HBAGetEventBufferFunc)(HBA_HANDLE, PHBA_EVENTINFO, HBA_UINT32
*);
typedef HBA_STATUS      (* HBASetRNIDMgmtInfoFunc) (HBA_HANDLE, PHBA_MGMTINFO);
typedef HBA_STATUS      (* HBAGetRNIDMgmtInfoFunc)(HBA_HANDLE, PHBA_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);


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;
```

5 HBA Common API

5.1 Overview

The following functions are available in the API for managing a Fibre Channel HBA. This API would be exposed to upper level applications to control SAN functionality.

5.1.1 General Vendor-Implemented Functions

```
HBA_STATUS HBA_RegisterLibrary(PHBA_ENTRYPOINTS entrypoints);

HBA_UINT32 HBA_GetVersion();

HBA_STATUS HBA_LoadLibrary();

HBA_STATUS HBA_FreeLibrary();

HBA_UINT32 HBA_GetNumberOfAdapters();

HBA_STATUS HBA_GetAdapterName(HBA_UINT32 adapterindex, char *adaptername);

HBA_HANDLE HBA_OpenAdapter(
       char* adaptername
       );

void HBA_CloseAdapter(
       HBA_HANDLE handle
       );
```

5.1.2 Informational Functions

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

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

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

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

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

5.1.3 FC-3 Management Functions

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

HBA_STATUS HBA_GetEventBuffer(
       HBA_HANDLE handle,
       PHBA_EVENTINFO EventBuffer,
       HBA_UINT32 *EventCount);


HBA_API HBA_STATUS HBA_SetRNIDMgmtInfo(
       HBA_HANDLE handle,
       HBA_MGMTINFO *pInfo);

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

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

5.1.4 FCP Information Functions

```
HBA_STATUS HBA_GetFcpTargetMapping (
```

```
        HBA_HANDLE handle,
        PHBA_FCPTARGETMAPPING mapping;
        );

HBA_STATUS HBA_GetFcpPersistentBinding (
        HBA_HANDLE handle,
        PHBA_FCPBINDING binding
        );
```

5.1.5 SCSI Information Functions

```
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);

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

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

5.1.6 Control Functions

```
void HBA_RefreshInformation(HBA_HANDLE handle);

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

6 HBA Common API Reference

6.1 HBA_UINT32 HBA_GetVersion();

Description

Returns the version which the common HBA API library is compatible with.

Arguments

None.

Return Values

A return value of 1 indicates version 1 of this API. No other return value is currently valid.

Example

```
HBA_UINT32 version;

version = HBA_GetVersion();

printf("Running version %d of the HBA API library.", version);
```

6.2 HBA_STATUS HBA_LoadLibrary();

Description

Loads the HBA Library. Must be called before calling any HBA library functions.

Arguments

None.

Return Values

A return value of HBA_STATUS_OK indicates the library loaded properly. A return value of HBA_STATUS_ERROR indicates a problem with loading.

Example

```
HBA_STATUS status;

status = HBA_LoadLibrary();

printf("Successfully loaded HBA library.\n");
```

6.3 HBA_STATUS HBA_FreeLibrary();

Description

Frees the HBA Library. Must be called after all HBA library functions to free all resources.

Arguments

None.

Return Values

A return value of HBA_STATUS_OK indicates the library was able to free all resources. A return value of HBA_STATUS_ERROR indicates a problem with freeing resources.

Example

```
HBA_STATUS status;

status = HBA_FreeLibrary();

printf("Successfully freed  HBA library.\n");
```

6.4 HBA_STATUS HBA_RegisterLibrary(PHBA_ENTRYPOINTS HBAInfo);

Description

Registers a specific vendor implementation of this library with the common API. This is only implemented for a vendor library and called by the Common HBA API layer.

Arguments

HBAInfo.

Return Values

A library should return HBA_STATUS_OK for calls to this function.

Example

```
/* 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;
```

6.5 HBA_UINT32 HBA_GetNumberOfAdapters();

Description

Returns the number of HBAs supported by the library. This returns the current number of HBAs, even if this changes.

Arguments

None.

Return Values

This function returns the number of adapters supported by this library. If no adapters are supported, the library should return 0.

Example

```
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);
       }
)
```

*6.6 HBA_STATUS HBA_GetAdapterName(UINT32 adapterindex, char *adaptername);*

Description

Returns the text string which describes this adapter and which is used to open the adapter with the library.

Arguments

adapterindex	The index to which adapter to retrieve the name.
adaptername	A text description, used to open an adapter as well as for a human-readable identification of an adapter instance in the form: mfg-model-adapterindex Examples: qlogic-qla2200-1 troika-zentai-1 emulex-lp8000-1 agilent-tachlite-1 jni-emerald-1

Return Values

This function returns HBA_STATUS_OK on success, and returns the string corresponding to the index of adapter in *adaptername*.

Example

```
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);
       }
)
```

6.7 HBA_HANDLE HBA_OpenAdapter(char adaptername);*

Description

Opens a named adapter. By opening an adapter, an upper level application is ensuring that all access to an HBA_HANDLE between an open and a close is to the same adapter. An HBA_OpenAdapter does not necessarily imply a driver "open", which is vendor implementation dependent.

Arguments

adaptername	A text description of an adapter as retrieved from HBA_GetAdapterName.

Return Values

This function returns a valid HBA_HANDLE on success, 0 on failure.

Example

```
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);
              }
       }
)
```

6.8 void HBA_CloseAdapter(HBA_HANDLE handle);

Description

Closes an open adapter.

Arguments

handle HBA_HANDLE to a previously opened adapter.

Return Values

None.

Example

```
adapterhandle = HBA_OpenAdapter(adaptername);
if (adapterhandle != NULL) {
       printf("Successfully opened %s\r\n",adaptername);
       HBA_CloseAdapter(adapterhandle);
}
```

6.9 HBA_STATUS HBA_GetAdapterAttributes(HBA_HANDLE handle, PHBA_ADAPTERATTRIBUTES adapterattributes);

Description

Retrieves the attributes for an adapter.

Arguments

handle HBA_HANDLE to a previously opened adapter.

Return Values

HBA_GetAdapterAttributes returns HBA_STATUS_OK if it is able to retrieve the attributes of an adapter.

HBA_ADAPTERATTRIBUTES includes:

Manufacturer
SerialNumber
Model
ModelDescription
NodeWWN
NodeSymbolicName
HardwareVersion
DriverVersion
OptionROMVersion
FirmwareVersion
VendorSpecificID
NumberOfPorts
DriverName

Example

```
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);
```

*6.10 HBA_STATUS HBA_GetAdapterPortAttributes(HBA_HANDLE handle, HBA_UINT32 portindex, HBA_PORTATTRIBUTES *portattributes);*

Description

Retrieves the attributes for a specified port on an adapter.

Arguments

handle	HBA_HANDLE to a previously opened adapter.
portindex	index of the port to query.

Return Values

HBA_GetAdapterPortAttributes returns HBA_STATUS_OK if it is able to retrieve the attributes of a port on an adapter.

HBA_PORTATTRIBUTES includes:

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

Example

```
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 a Port FcID of %d",
                     i, portattributes.PortFcId);
       }
}
```

*6.11 HBA_STATUS HBA_GetPortStatistics(HBA_HANDLE handle, HBA_UINT32 portindex, HBA_PORTSTATISTICS *portstatistics);*

Description

Retrieves the statistics for a specified port on an adapter.

Arguments

handle	HBA_HANDLE to a previously opened adapter.
portindex	index of the port to query.

Return Values

HBA_GetPortStatistics returns HBA_STATUS_OK if it is able to retrieve the statistics of a port on an adapter. If an HBA does not support a specific statistic it should return return an All-Ones unsigned integer.

HBA_PORTSTATISTICS includes:

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

Example

```
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);
      }
}
```

*6.12 HBA_STATUS HBA_GetDiscoveredPortAttributes(HBA_HANDLE handle, HBA_UINT32 portindex, HBA_UINT32 discoveredportindex, HBA_PORTATTRIBUTES *portattributes);*

Description

Retrieves the attributes for a specified port discovered in the network.

Arguments

handle	HBA_HANDLE to a previously opened adapter.
portindex	index of the port to query.
discoveredportindex	index of the discovered port to query

Return Values

HBA_GetDiscoveredPortAttributes returns HBA_STATUS_OK if it is able to retrieve the attributes of a port discovered on a network.

HBA_PORTATTRIBUTES includes:

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

In the case of HBA_GetDiscoveredPortAttributes NumberOfDiscoveredPorts is always 0.

Example

```
HBA_STATUS status;
HBA_PORTATTRIBUTES portattributes;

/* Get the attributes for the first discovered port
on first adapter port */
status = HBA_GetDiscoveredPortAttributes(handle, 1, 1, &portattributes);
if (status == HBA_STATUS_OK) {
       printf("Port 1 on Adapter Port 1 has a Port FcID of %d",
                     portattributes.PortFcId);
}
```

*6.13 HBA_STATUS HBA_GetPortAttributesByWWN(HBA_HANDLE handle, HBA_WWN PortWWN, HBA_PORTATTRIBUTES *portattributes);*

Description

Retrieves the attributes for a specific discovered port by WWN.

Arguments

handle	HBA_HANDLE to a previously opened adapter.
PortWWN	WWN of the port to find.
portattributes	HBA_PORTATTRIBUTES structure to fill in.

Return Values

HBA_GetPortAttributesByWWN returns HBA_STATUS_OK if it is able to retrieve the attributes of a port given its Port WWN.

Example

6.14 void HBA_RefreshInformation(HBA_HANDLE handle);

Description

Refreshes information about an HBA.

Arguments

handle	Handle to an open HBA.

Return Values

None.

Example

```
HBA_RefreshInformation(adapterhandle);
status = HBA_GetPortStatistics(adapterhandle, portindex,
             &portstats);
```

6.15 void HBA_ResetStatistics(HBA_HANDLE handle, HBA_UINT32 portindex);

Description

Reset statistics information on an HBA.

Arguments

handle	Handle to an open HBA.

Return Values

None.

Example

```
HBA_ResetStatistics(adapterhandle, portindex);
printf("Reset statistics for adapter %d", portindex);
```

6.16 HBA_STATUS HBA_GetFcpTargetMapping(HBA_HANDLE handle, PHBA_FCPTARGETMAPPING mapping);

Description

Retrieves the mapping between FCP targets and OS SCSI information.

Arguments

handle	Handle to an open HBA.
mapping	Pointer to an HBA_FCPTARGETMAPPING structure. The size of this structure is dependent on the NumberOfEntries value within the structure, and can be of arbitrary size. An upper level application can 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.

Return Values

Return is HBA_STATUS_OK if the HBA is able to retrieve the information. mapping contains the full mapping information of local SCSI LUNs to FCP LUNs for this HBA.

The value of the NumberOfEntries field of the returned structure will 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 them.

Example

6.17 HBA_STATUS HBA_GetFcpPersistentBinding(HBA_HANDLE handle, PHBA_FCPBINDING binding);

Description

Get persistent bindings between an FCP target and a SCSI ID.

Arguments

handle	Handle to an open HBA.
binding	Pointer to a HBA_FCPBINDING structure. The size of this structure is dependent on the NumberOfEntries value within the structure, and can be of arbitrary size. An upper level application can 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.

Return Values

Return is HBA_STATUS_OK if the HBA is able to retrieve the information. binding contains the full binding information for of local SCSI LUNs to FCP LUNs for this HBA.

The value of the NumberOfEntries field of the returned structure will 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 them.

Example

*6.18 HBA_STATUS HBA_GetEventBuffer(HBA_HANDLE handle, PHBA_EVENTINFO EventBuffer, HBA_UINT32 *EventCount);*

Description

Remove and return the next events from the HBA's event queue. The number of events returned will be the lesser of the value of argument EventBufferSize at call and the number of entries available in the event queue.

This provides a simple polled interface to a basic set of HBA-detected events. An improved method of notification is intended for an early subsequent version of this API.

The event queue internal implementation is not constrained but its behavior is a circular queue of event records (structure HBA_EVENTINFO) which represent RSCN, link status, or other events. Event records are added as events occur and removed in order of occurrence as any application gets them. The size of the queue is implementation dependent. If the queue becomes full, any newly added records will replace the oldest records, causing the oldest records to be lost. If multiple applications make overlapping sequences of HBA_GetEventBuffer calls, the available events will each be delivered to only one of the applications. The exact distribution is not predictable, but the sequence of events delivered to any application will still be strictly in order of event occurrence.

NOTE: The arrival of an RSCN ELS will be treated as a separate event for each "Affected N-Port ID Page" carried by the RSCN.

Arguments

handle	Handle to an open HBA.
EventBuffer	Pointer to a buffer to receive events.
EventCount	Number of event records in the buffer to receive events. Set to the size (in event records) of the buffer for receiving events on call, and returned as the number of events actually delivered.

Return Values

Return is HBA_STATUS_OK if the HBA is able to retrieve the information.

Example

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

Description

Send a CT passthrough frame. An HBA should decode this CT_IU request per the FS-GS3 specification, routing the CT frame in a fabric according to the GS_TYPE field within the CT frame.

Arguments

handle	Handle to an open HBA.
pReqBuffer	Pointer to a buffer containing the full CT frame.
ReqBufferSize	Size of the buffer of the send buffer in bytes.
pRspBuffer	Pointer to a buffer containing the received CT frame.
RspBufferSize	Size of the buffer for the received CT frame in bytes.

Return Values

Return is HBA_STATUS_OK if the HBA is able to execute the command.

Example

6.20 HBA_STATUS HBA_SetRNIDMgmtInfo(HBA_HANDLE handle, HBA_MGMTINFO info);

Description

Sets the RNID (Request Node Identification Information Data) returned from the HBA.

Arguments

handle	Handle to an open HBA.
info	Management information.

Buffer Offset	Size (Bytes)	Description (Big Endian Format)
0	16	WWN (left justified)
16	4	Unit Type
20	4	Port ID
24	4	Number of Attached Nodes
28	2	IP Version
30	2	UDP Port Number
32	16	IP Address
48	2	Reserved
50	2	Topology Discovery Flags
	52	Total Request Size in bytes

Return Values

Return is HBA_STATUS_OK if the HBA is able to set the information.

Example

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

Description

Returns the RNID (Request Node Identification Information Data) from the HBA.

Arguments

handle	Handle to an open HBA.
pInfo	Pointer to buffer in Big Endian format.

Return Values

Return is HBA_STATUS_OK if the HBA is able to retrieve the information.

The response buffer has the following format:

Buffer Offset	Size (Bytes)	Description (Big Endian Format)
0	16	WWN (left justified)
16	4	Unit Type
20	4	Port ID
24	4	Number of Attached Nodes
28	2	IP Version
30	2	UDP Port Number
32	16	IP Address
48	2	Reserved
50	2	Topology Discovery Flags
	52	Total Request Size in bytes

Example

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

Description

Issues an ELS RNID (Request Node Identification Data) to another node.

Arguments

handle	Handle to an open HBA.
wwn	Fabric address of a device's name
wwntype	The type of this address.
pRspBuffer	Response buffer in Big Endian format.
RspBufferSize	On call, set to length of pRspBuffer. On return, returns the size of data written to pRspBuffer.

Return Values

Return is HBA_STATUS_OK if the HBA is able to retrieve the information.

Response Buffer Format:

Buffer Offset	Size (Bytes)	Description (Big Endian Format)
0	1	Node Identification Data Format
1	1	Common Node Identification Data Length (0 or 16)
2	1	Reserved
3	1	Specific Node Identification Data Length = 52
4	0 or 16	Common Node Identification Data
4 or 20	52	Specific Node Identification Data
	56 or 72	Total Request Size in bytes

When the Common Node Identification Data Length indicates that Common Node Identification Data exists, the field shall contain the unit's Worldwide Names (Port Name and Node Name) as follows:

Size in Bytes	Format (Big Endian)
8	Port Name
8	Node Name

The Specific Node Identification Data for DataFmt = DFh (Topology Discovery) is as follows:

Size in Bytes	Format (Big Endian)
16	Global ID (left justified)
4	Unit Type
4	Physical Port Number
4	Number of Attached Nodes
2	IP Version
2	UDP Port Number
16	IP Address
2	Reserved
2	Topology Discovery Flags

Unit Type is defined as follows:

Value (hex)	Type (Big Endian)
00000001	Unknown
00000002	Other (none of the following)
00000003	Hub
00000004	Switch
00000005	Gateway
00000006	Converter
00000007	HBA
00000008	Proxy-agent
00000009	Storage Device (disk, CD, tape, etc)
0000000A	Host
0000000B	Storage subsystem (raid, library, etc)
0000000C	Module (subcomponent of a system)
0000000D	Software Driver
00000000 0000000E through FFFFFFFF	Reserved

IP Version is defined as follows:

Value (hex)	Version Description (Big Endian)
0000	None
0001	IP (IP version 4)
0002	IP6 (IP version 6)
3 through FFFF	Reserved

Topology Discovery Flags are defined as follows:

Bit 0 Topology discovery Support (T): When set, the node supports further topology-discovery inquires.
Bit 1 Loop Position Valid (L): When set and multiple nodes are reported, the value signals that the Node Identification Data records reported are in the order detected from the successful execution of a Loop Initialization Report Primitive.

*6.23 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);*

Description

Sends a SCSI inquiry to a remote WWN.

Arguments

handle	Handle to an open HBA.
PortWWN	Port WWN of an HBA
fcLUN	Specific FC Lun to send the inquiry to.
EVPD	Set to 0 to return the standard SCSI INQUIRY data. Set to 1 to return the vital product data specified by the page code.
PageCode	If EVPD is 1, the Vital Product Data page code to request.
pRspBuffer	Pointer to a buffer to receive the response
RspBufferSize	Size of the buffer to receive response
pSenseBuffer	Pointer to buffer to receive sense data
SenseBufferSize	Size of the buffer to receive sense information

Return Values

pRspBuffer	Contains the response to the inquiry
pSenseBuffer	Contains the sense data for the command

Example

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

Description

Sends a SCSI report luns command to a remote WWN.

Arguments

handle	Handle to an open HBA.
PortWWN	Port WWN of an HBA
pRspBuffer	Pointer to a buffer to receive the response
RspBufferSize	Size of the buffer to receive response
pSenseBuffer	Pointer to buffer to receive sense data
SenseBufferSize	Size of the buffer to receive sense information

Return Values

pRspBuffer	Contains the response to the report luns command
pSenseBuffer	Contains the sense data for the command

Example

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

Description

Sends a read capacity to a remote WWN.

Arguments

handle	Handle to an open HBA.
PortWWN	Port WWN of an HBA
fcLUN	Specific FC Lun to send the read capacity to.
pRspBuffer	Pointer to a buffer to receive the response
RspBufferSize	Size of the buffer to receive response
pSenseBuffer	Pointer to buffer to receive sense data
SenseBufferSize	Size of the buffer to receive sense information

Return Values

pRspBuffer	Contains the response to the read capacity command
pSenseBuffer	Contains the sense data for the command

Example

7 Appendix A: Multi Vendor Interoperability

An important consideration in implementing this API is to support a single API to query common information from multiple adapters manufactured by different vendors are operating within a single server.

7.1 Win32

In a Win32 environment (Window NT, Windows 2000) the method for registering multiple vendors is the following:

Under the Registry, an HBA vendor will install a registry key to indicate where the vendor library is installed.

\\HKEY_LOCAL_MACHINE\SOFTWARE\SNIA\HBA*vendorid*

A value named LibraryFile of type REG_SZ will contain the full path to the vendor's library

vendorid is an arbitrary value which uniquely identifies the vendor library.

Example:
\\HKEY_LOCAL_MACHINE\SOFTWARE\SNIA\HBA\samplevendor
LibraryFile = "c:/Program Files/Samplevendor/Library.dll"

The following method will be used to load multiple vendors' libraries:

1. A HBA API "wrapper" will read the registry for library names
2. Using the Win32 routines LoadLibrary and GetProcAddress, the wrapper will open and discover the appropriate vendors libraries.
3. The HBA API wrapper will use these libraries to discover the aggregate number of adapters and report this to the upper level application.
4. The names of the lower level adapters will be passed through the HBA API wrapper.
5. A call to open an adapter will be "switched" by the HBA API wrapper, which will use the reserved upper 16 bits of the HBA_HANDLE to determine which adapter to address on a given routine.
6. Remaining calls will be routed by the HBA API wrapper to the appropriate library given the HBA_HANDLE.

7.2 Unix

In a Unix environment (Window NT, Windows 2000) the method for registering multiple vendors is the following:

/etc/hba.conf contains the following:

```
#
# This file contains names and references to HBA libraries
#
# Format:
#
# <library name>  <library pathname>
#
# The space between the name and path are
#
# For example:
#
#   tachlite        c:\system32\drivers\hba\tachlite.dll
#   ql2x00          c:\Program Files\QLogic Corporation\QSDMGT\Lib\QSDMGT.DLL
#   troika          c:\system32\drivers\ hba\troika.dll
#   fci1063         c:\system32\drivers\ hba\fci1063.dll
```

lp8000 c:\system32\drivers\hba\lp8000.dll

The Unix method for loading vendor libraries will be identical to the Win32 method, with the following difference:

1. The routines for library loading will differ from Unix version-to-version. In the case of Solaris, the routines are dlopen and dlsym.

8 Appendix B: Naming, Handles and Their Usage

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

A name is:

- Unique to an instance of an adapter
- Identifier for adapters you can open and manage
- **Not** guaranteed to be the same across reboots

A handle is:

- Persistent between and open and a close of an adapter
- For an individual vendor the lower 16 bits are determined entirely by the vendor
- Upper 16 bits are reserved for use by the HBA wrapper library
- Related on a one-to-one basis between a specific instance of an HBA and a handle
- **Not** guaranteed to be the same across reboots
- **Not** guaranteed to be the same across opens

The following situations require vendor-specific handling to properly map adapters:

- Addition of a new HBA
- An instance of an HBA being removed
- An HBA being replaced

Addition of a new HBA
In the case of the addition of a new HBA, a new name should be assigned to the device which does not conflict with previous names.

An instance of an HBA being removed
When an HBA is removed, it is suggested that the name of the device should not generally be re-used, in order for upper level software to reliably reference the same device.

An HBA being replaced
In the case of an HBA being replaced, the functionality is up to the vendor's implementation. If an HBA is replaced and there is no change in functionality, WWN, or any other HBA properties the same name is appropriate. Any change in the properties, including WWN, should result in a new adapter being instantiated.

9 Appendix C: Future Enhancements

This appendix lists possible future enhancements for this spec.

Asynchronous Event Notification
Kernel mode operation
Control ability
Ability to initiate LIP
Ability to send low level ELS frames
Expansion of Management Server functions
Ability to do a target reset
Ability to send raw FCP-SCSI frames
Writeable RSCN information
Fabric login parameters
E_D_TOV
R_A_TOV
Connected media for a port
Loop position map for a port
I/Os for a port, on a per FC-4 basis
(and on a per LUN basis, for FCP)
Megabytes transferred for a port, on a per FC-4 basis
(and on a per LUN basis, for FCP)

10 References

"Toward Binary Compatibility of VIA Implementations on WIN32", Jim Lyon, Microsoft Corp.
T11/98-435v1 FC-GS-2 rev 5.3

"Qlogic SAN/Device Management API" Draft Version 1.3 05/26/00

"Fibre Alliance HBA Application Programming Interface Specification" Version 1.0 06/02/00