SM-HBA:
SAS/FC Common HBA API

T10/07-221r0

3 May 2007

Bob Nixon/Emulex

Introduction to SM-HBA

- A work undertaken in T11
  - Thanks to Krithivas (Intel) for editing and most of the technical work
- Major goals
  - Add an API for SAS HBAs
  - Include the richer port structure of SAS
  - Re-model the FC API to maximize common API for common features
- Now ANSI/INCITS 428/2007

- A word of apology: the following is written in pidgin-UML (no that does not mean it is sprinkled with white splats)

SM-API context (with some lapses)

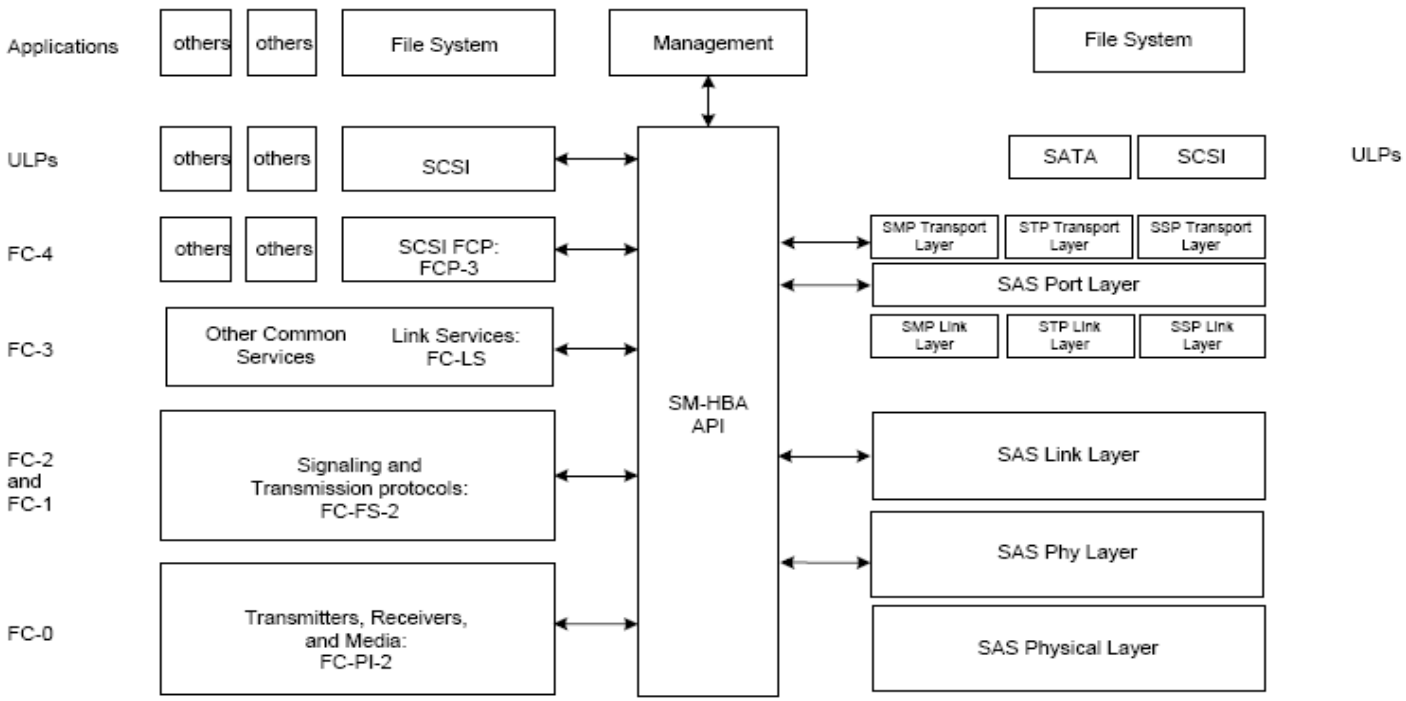
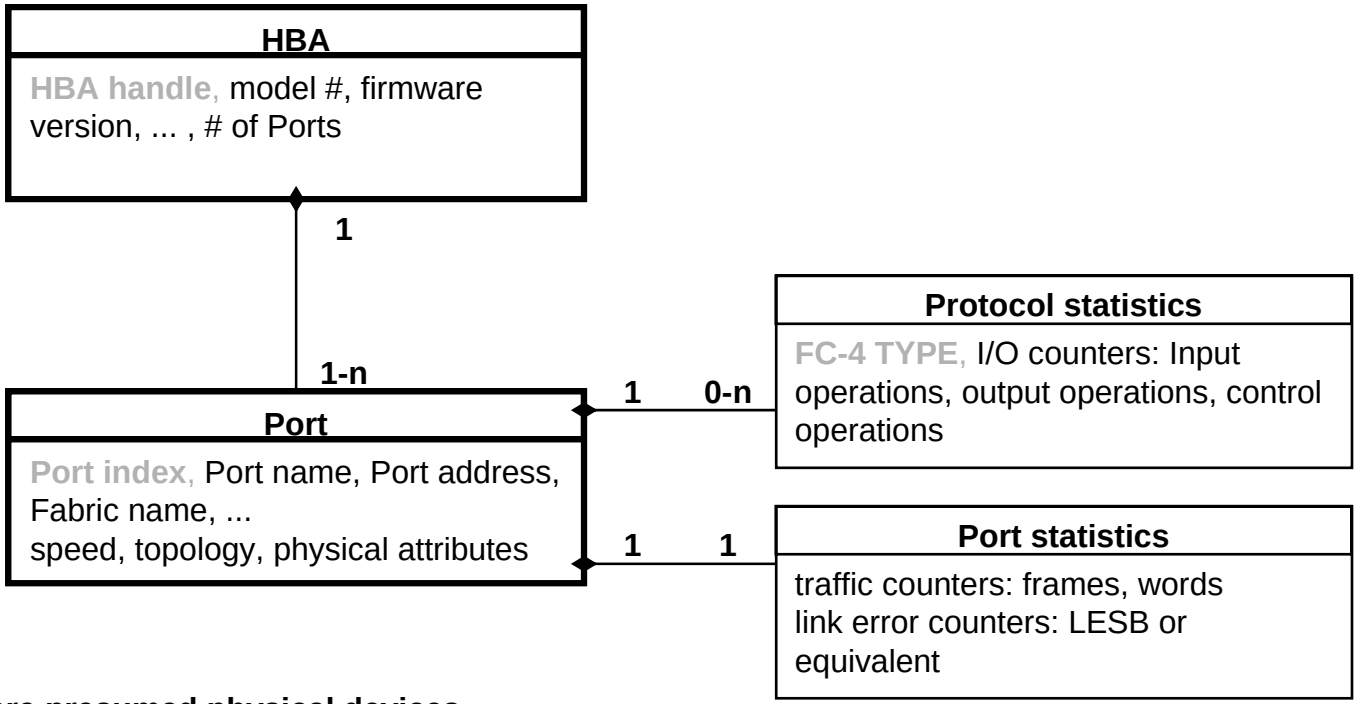


Figure 1 - Context for SM-HBA

FC-HBA “model”



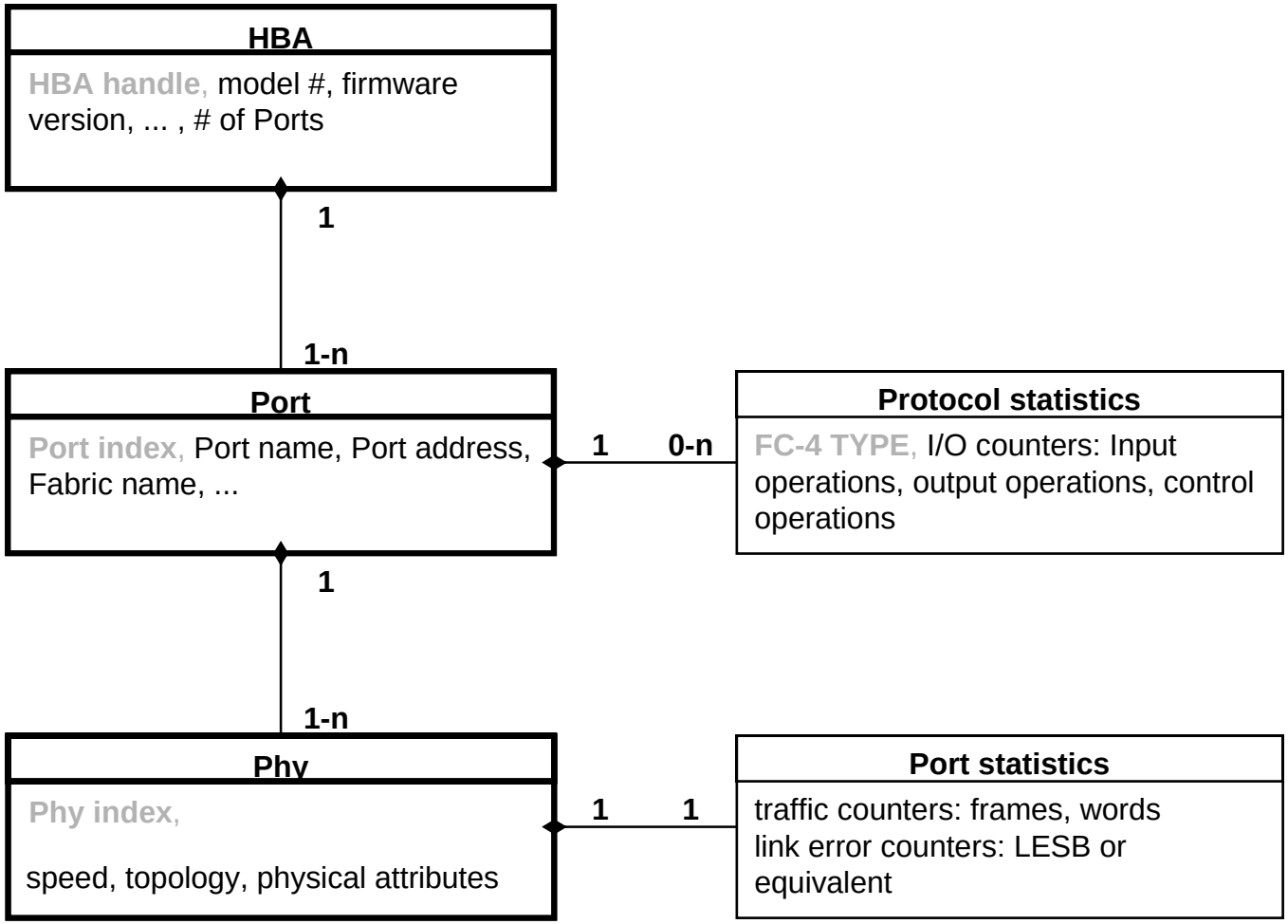
heavy boxes are presumed physical devices

Grey text **denotes implicit attributes, typically used to identify an object within some scope**

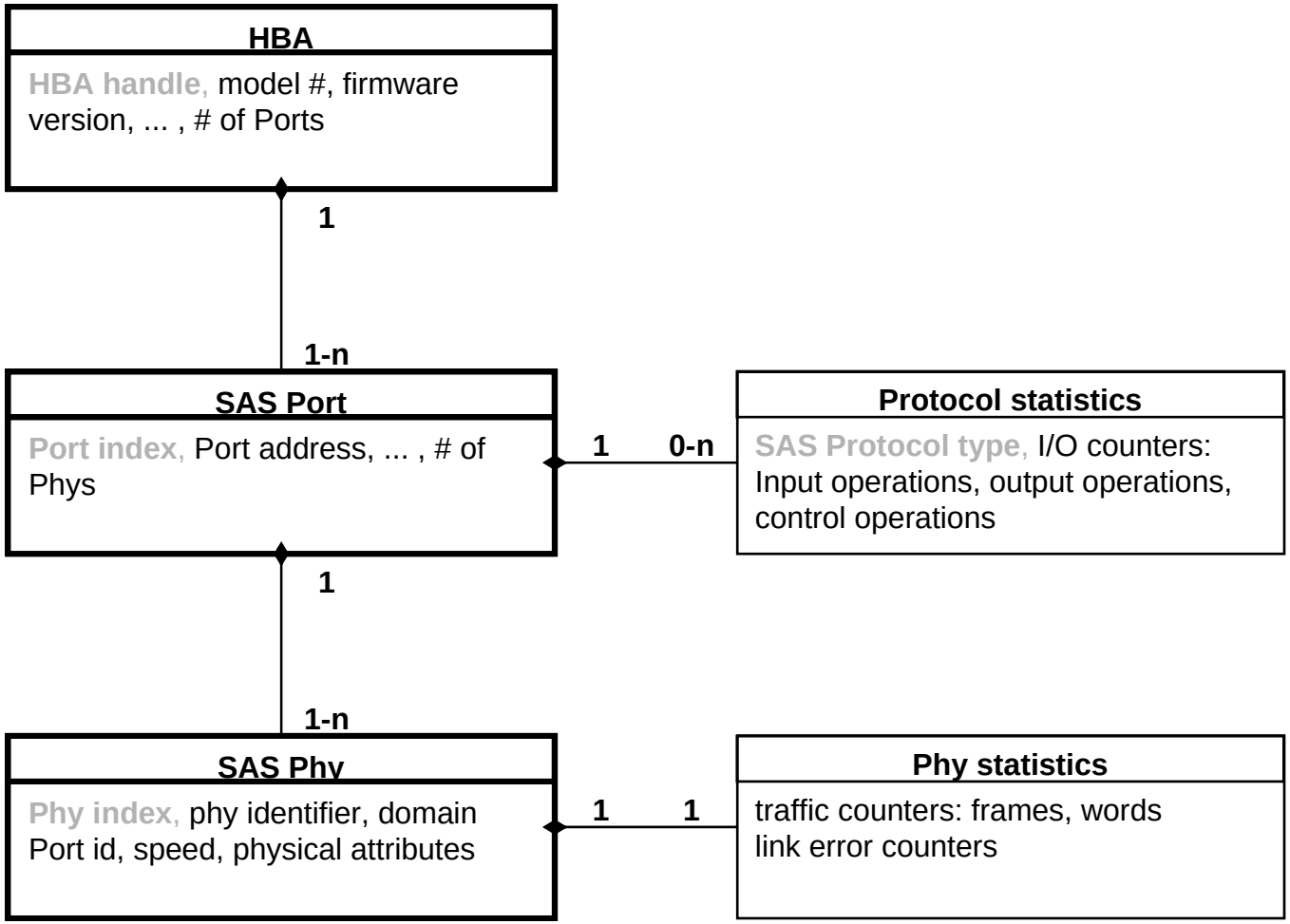
Green stuff is added complic ... ah ... components

Pink text **is just to draw your attention**

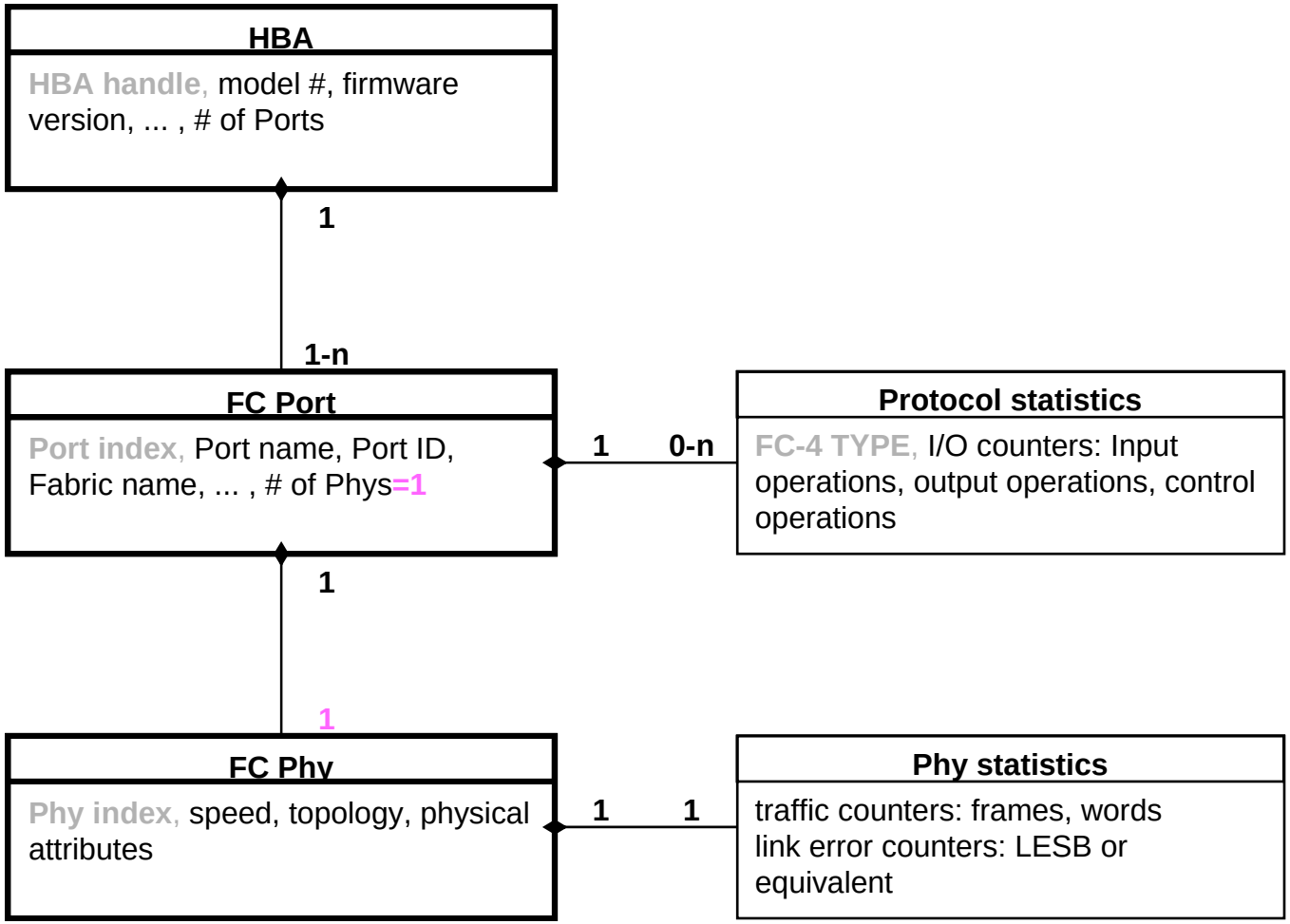
SM-HBA “model” grew from FC-HBA “model”



SM-HBA structure: A SAS HBA



SM-HBA structure: An FC HBA



API for Adapter configuration

- Pretty much identical for SAS, FC, and mixed HBAs
- Hardware identification and inventory information
- A port count

API for Port configuration

- Not quite identical for SAS and FC
- Hardware identification and inventory information
- Port name/address information
  - WWPN for both
  - N_Port_ID for FC
- Discovered Port count
- Port type (FC or SAS)
- Type-specific protocol support (e.g., FCP, SSP)
- Phy count (for FC, it's restricted to small values of 1)

API for Phy configuration

- Mostly divergent for SAS and FC
- Hardware identification
- Type-specific speeds supported and active
- Max frame size for FC
- Domain ID for SAS

API for statistics

- Structurally the same
  - Protocol statistics per Port
  - Link statistics per Phy
- But different statistics for SAS than FC

API for discovered device management

- Manages HBAs/drivers that map SAN devices into a virtual SPI interface (bus, target, LUN)
- Very similar…For each local port:
  - List of discovered ports, with limited port info for each
    - May include switch/expander ports (e.g., FC Name Server; SAS SMP)
  - List of SCSI devices/logical units currently mapped into the OS (called “target mappings”, though they probably are LU mappings)
  - List of SCSI devices/logical units that are desired to be mapped into the OS (called “persistent bindings”). This is one of the few configurable entities

API for Fabric/Domain management

- Management command passthrough functions
  - Passthrough of application-constructed SMP IU for SAS
  - Passthrough of application-constructed CT IU for FC
- Specific functions to construct certain FC ELSs
  - Parameters for specific ELS fields, not just a raw IU
- All send one command, wait for a response, and the raw response IU is among the function return parameters
- We’re a little nervous how much damage might be done by the passthrough functions

API for SCSI discovery

- If we were nervous about management passthrough, the idea of SCSI passthrough REALLY bothered us
  - Our OS friends made sure we remained appropriately concerned
  - Eventually we set our ground rule: we would pass through no SCSI command that was not essential to discovery
  - These are not raw passthrough. Like the FC ELS functions, they have parameters for each field.
  - This led to…
- Three SCSI command APIs:
  - REPORT LUNS
  - INQUIRY
  - READ CAPACITY
- Identical for FC and SAS

API for asynchronous event notification

- Same architecture, similar event classes, some common events
- API registers callback functions for any of several event classes
  - Registrations remain until explicitly removed (or the application goes away).
  - May register multiple callbacks for a class; they all get called for each occurrence. Caveat emptor.
- Callback function is called when an event in its class occurs
  - Callback function receives event identity and a “token” from the registration call
  - The token allows common handling of several event classes

What's in SM-HBA-2

- API for FC virtualization features 
- Functions to create relationships among ports and phys
  - May apply to reconfiguring port/phy assignment in SAS
- A place to keep host bus parameters (e.g., PCI address)
- API for SCSI Management Protocol operations
- Time and interest permitting, API for management of host security policy (note, this is not aimed at zoning, that's considered fabric policy)
  - SCSI key management would be within scope
- More flexible API for searching configuration (e.g., traverse relationships both ways, not just top-down)

✓

Questions?

✓

Backup material

API for Adapter configuration

6.3.1 Generic Adapter Attribute

Common Adapter

```
typedef struct SMHBA_AdapterAttributes {
        char                    Manufacturer[64];
        char                    SerialNumber[64];
        char                    Model[256];
        char                    ModelDescription[256];
        char                    HardwareVersion[256];
        char                    DriverVersion[256];
        char                    OptionROMVersion[256];
        char                    FirmwareVersion[256];
        HBA_UINT32              VendorSpecificID;
        char                    DriverName[256];
        char                    HBASymbolicName[256];
        char                    RedundantOptionROMVersion[256];
        char                    RedundantFirmwareVersion[256];
} SMHBA_ADAPTERATTRIBUTES, *PSMHBA_ADAPTERATTRIBUTES;
```

API for Port configuration

```
typedef struct SMHBA_PortAttributes {
        HBA_PORTTYPE            PortType;
        HBA_PORTSTATE           PortState;
        char                    OSDeviceName[256];
        SMHBA_PORT              PortSpecificAttribute;
} SMHBA_PORTATTRIBUTES, *PSMHBA_PORTATTRIBUTES;
```

Common Port

```
typedef struct SMHBA_FC_Port {
        HBA_WWN                 NodeWWN;
        HBA_WWN                 PortWWN;
        HBA_UINT32              FcId;
        HBA_COS                 PortSupportedClassofService;
        HBA_FC4TYPES            PortSupportedFc4Types;
        HBA_FC4TYPES            PortActiveFc4Types;
        HBA_WWN                 FabricName;
        char                    PortSymbolicName[256];
        HBA_UINT32              NumberofDiscoveredPorts;
        HBA_UINT8               NumberofPhys;
}SMHBA_FC_PORT, *PSMHBA_FC_PORT;
```

FC Port

```
typedef struct SMHBA_SAS_Port {
        HBA_SASPORTPROTOCOL         PortProtocol;
        HBA_WWN                     LocalSASAddress;
        HBA_WWN                     AttachedSASAddress;
        HBA_UINT32                  NumberofDiscoveredPorts;
        HBA_UINT32                  NumberofPhys;
} SMHBA_SAS_PORT, *PSMHBA_SAS_PORT;
```

SAS Port

API for Phy configuration

SAS Phy

```
typedef struct SMHBA_SAS_Phy {
                HBA_UINT8                   PhyIdentifier;
                HBA_SASPHYSPEED             NegotiatedLinkRate;
                HBA_SASPHYSPEED             ProgrammedMinLinkRate;
                HBA_SASPHYSPEED             HardwareMinLinkRate;
                HBA_SASPHYSPEED             ProgrammedMaxLinkRate;
                HBA_SASPHYSPEED             HardwareMaxLinkRate;
                HBA_WWN                     domainPortWWN;
} SMHBA_SAS_PHY, *PSMHBA_SAS_PHY;
```

FC Phy

```
typedef struct SMHBA_FC_Phy {
       HBA_FCPHYSPEED       PhySupportedSpeed;  /* PhySupportedSpeed */
       HBA_FCPHYSPEED       PhySpeed;           /* PhySpeed */
       HBA_FCPHYTYPE        PhyType;
       HBA_UINT32           MaxFrameSize;        /* MaxFrameSize */
} SMHBA_FC_PHY, *PSMHBA_FC_PHY;
```

API for statistics

Protocol Statistics

```
/* Statistical counters for FC-4, SSP, STP, SMP protocols */
typedef struct SMHBA_ProtocolStatistics {
        HBA_INT64           SecondsSinceLastReset;
        HBA_INT64           InputRequests;
        HBA_INT64           OutputRequests;
        HBA_INT64           ControlRequests;
        HBA_INT64           InputMegabytes;
        HBA_INT64           OutputMegabytes;
} SMHBA_PROTOCOLSTATISTICS, *PSMHBA_PROTOCOLSTATISTICS;
```

SAS phy statistics

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

FC phy statistics

```
typedef struct SMHBA_FCPhyStatistics {
        HBA_INT64           SecondsSinceLastReset;
        HBA_INT64           TxFrames;
        HBA_INT64           TxWords;
        HBA_INT64           RxFrames;
        HBA_INT64           RxWords;
        HBA_INT64           LIPCount;
        HBA_INT64           NOSCount;
        HBA_INT64           ErrorFrames;
        HBA_INT64           DumpedFrames;
        HBA_INT64           LinkFailureCount;
        HBA_INT64           LossOfSyncCount;
        HBA_INT64           LossOfSignalCount;
        HBA_INT64           PrimitiveSeqProtocolErrCount;
        HBA_INT64           InvalidTxWordCount;
        HBA_INT64           InvalidCRCCount;
}SMHBA_FCPHYSTATISTICS, *PSMHBA_FCPHYSTATISTICS;
```

API for discovered device management

Target Mapping entry

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

Target Mapping list

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

Persistent Binding entry

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

Persistent Binding list

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

API for Fabric/Domain management

Fabric and Domain Management Functions	
HBA_SendCTPassThruV2	8.4.1
HBA_SetRNIDMgmtInfo	8.4.2
HBA_GetRNIDMgmtInfo	8.4.3
HBA_SendRNIDV2	8.4.4
HBA_SendRPL	8.4.5
HBA_SendRPS	8.4.6
HBA_SendSRL	8.4.7
HBA_SendLIRR	8.4.8
HBA_SendRLS	8.4.9
SMHBA_SendTEST	8.4.10
SMHBA_SendECHO	8.4.11
SMHBA_SendSMPPassThru	8.4.12

API for Fabric/Domain management

```
HBA_UINT32 SMHBA_SendSMPPassThru(
        HBA_HANDLE    handle,
        HBA_WWN       hbaportWWN,
        HBA_WWN       destportWWN,
        HBA_WWN       domainPortWWN,
        void          *pReqBuffer,
        HBA_UINT32    ReqBufferSize,
        void          *pRspBuffer,
        HBA_UINT32    *pRspBufferSize
);
```

Example raw passthrough function

```
 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
 );
```

Example structured passthrough function

API for SCSI discovery

```
HBA_STATUS SMHBA_ScsiInquiry (
        HBA_HANDLE handle,
        HBA_WWN hbaPortWWN,
        HBA_WWN discoveredPortWWN,
        HBA_WWN domainPortWWN;
        SMHBA_SCSILUN smhbaLUN,
        HBA_UINT8 CDB_Byte1,
        HBA_UINT8 CDB_Byte2,
        void *pRspBuffer,
        HBA_UINT32 *pRspBufferSize,
        HBA_UINT8 *pScsiStatus,
        void *pSenseBuffer,
        HBA_UINT32 *pSenseBufferSize
);
```

Example SCSI passthru function

API for asynchronous event notification

Event Handling Functions	
SMHBA_RegisterForAdapterAddEvents	8.7.2
SMHBA_RegisterForAdapterEvents	8.7.3
SMHBA_RegisterForAdapterPortEvents	8.7.4
SMHBA_RegisterForAdapterPortStatEvents	8.7.5
SMHBA_RegisterForAdapterPhyStatEvents	8.7.6
SMHBA_RegisterForTargetEvents	8.7.7
HBA_RegisterForLinkEvents	8.7.8
HBA_RemoveCallback	8.7.9

8.7.4 SMHBA_RegisterForAdapterPortEvents

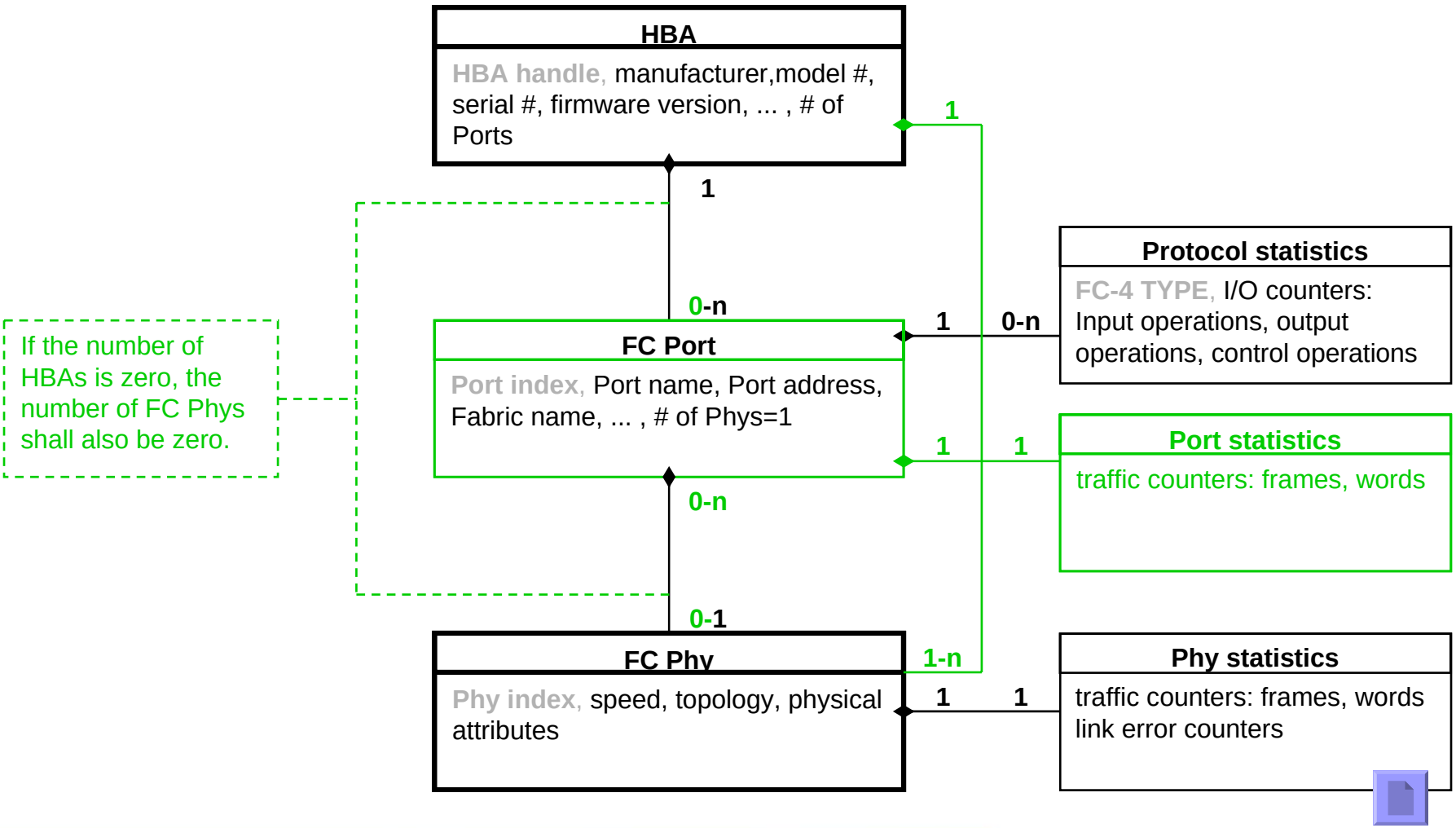
8.7.4.1 Format

```
HBA_STATUS SMHBA_RegisterForAdapterPortEvents(
       void (*pCallback) (
              void *pData,
              HBA_WWN portWWN,
              HBA_UINT32 eventType,
              HBA_UINT32 fabricPortID
       ),
       void *pUserData,
       HBA_HANDLE handle,
       HBA_WWN portWWN,
       HBA_UINT32 specificEventType,
       HBA_CALLBACKHANDLE *pCallbackHandle
);
```

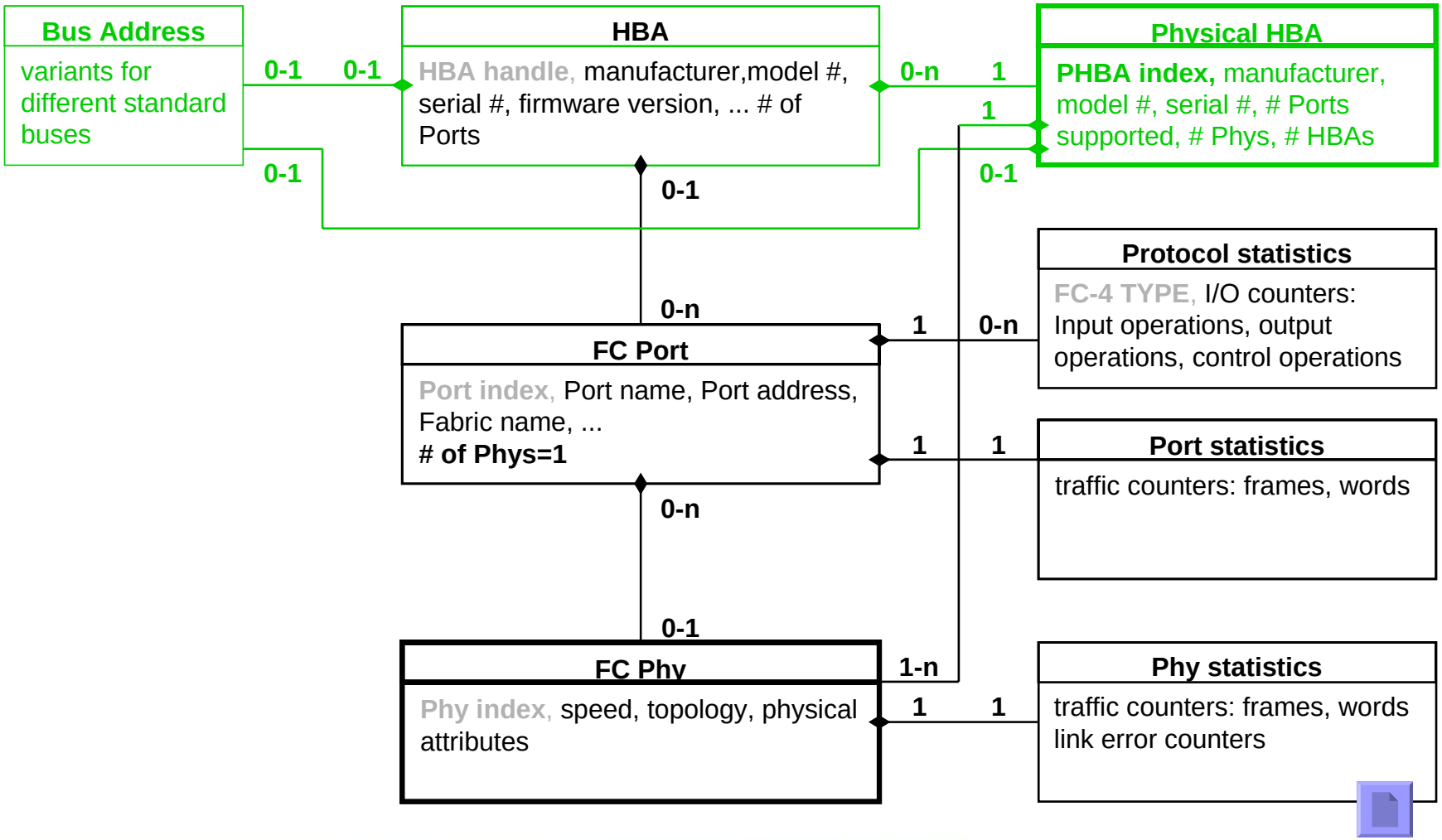
6.8.1.4 Port Category Event Types

```
#define     HBA_EVENT_PORT_UNKNOWN              0x200
#define     HBA_EVENT_PORT_OFFLINE              0x201
#define     HBA_EVENT_PORT_ONLINE               0x202
#define     HBA_EVENT_PORT_NEW_TARGETS          0x203
#define     HBA_EVENT_PORT_FABRIC               0x204
#define     HBA_EVENT_PORT_BROADCAST_CHANGE     0x205
#define     HBA_EVENT_PORT_BROADCAST_SES        0x208
#define     HBA_EVENT_PORT_BROADCAST_D24_0      0x206
#define     HBA_EVENT_PORT_BROADCAST_D27_4      0x207
#define     HBA_EVENT_PORT_BROADCAST_D01_4      0x209
#define     HBA_EVENT_PORT_BROADCAST_D04_7      0x20A
#define     HBA_EVENT_PORT_BROADCAST_D16_7      0x20B
#define     HBA_EVENT_PORT_BROADCAST_D29_7      0x20C
#define     HBA_EVENT_PORT_ALL                  0x2FF
```

Here's the trick for virtualizing FC Ports...



...a tweak more for the PCI IOV guys...



...and finally, VSANs