ISO/IEC 14165-521

Edition 1.0 2009-01

INTERNATIONAL STANDARD

Information technology – Fibre channel –
Part 521: Fabric application interface standard (FAIS)

ISO/IEC 14165-521:2009(E)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

- Catalogue of IEC publications: www.iec.ch/searchpub
The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub
Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv
If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

ISO/IEC 14165-521

Edition 1.0 2009-01

INTERNATIONAL STANDARD

Information technology – Fibre channel – Part 521: Fabric application interface standard (FAIS)

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE X

ICS 35.200

ISBN 978-2-88910-824-4

Contents **Page**

Figure **Page**

Table **Page**

INFORMATION TECHNOLOGY – FIBRE CHANNEL –

Part 521: Fabric application interface standard (FAIS)

FOREWORD

1) ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards. Their preparation is entrusted to technical committees; any ISO and IEC member body interested in the subject dealt with may participate in this preparatory work. International governmental and non-governmental organizations liaising with ISO and IEC also participate in this preparation.

2) In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

3) The formal decisions or agreements of IEC and ISO on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC and ISO member bodies.

4) IEC, ISO and ISO/IEC publications have the form of recommendations for international use and are accepted by IEC and ISO member bodies in that sense. While all reasonable efforts are made to ensure that the technical content of IEC, ISO and ISO/IEC publications is accurate, IEC or ISO cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

5) In order to promote international uniformity, IEC and ISO member bodies undertake to apply IEC, ISO and ISO/IEC publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any ISO/IEC publication and the corresponding national or regional publication should be clearly indicated in the latter.

6) ISO and IEC provide no marking procedure to indicate their approval and cannot be rendered responsible for any equipment declared to be in conformity with an ISO/IEC publication.

7) All users should ensure that they have the latest edition of this publication.

8) No liability shall attach to IEC or ISO or its directors, employees, servants or agents including individual experts and members of their technical committees and IEC or ISO member bodies for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication of, use of, or reliance upon, this ISO/IEC publication or any other IEC, ISO or ISO/IEC publications.

9) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

10) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 14165-521 was prepared by subcommittee 25: Interconnection of information technology equipment, of ISO/IEC joint technical committee 1: Information technology.

The list of all currently available parts of the ISO/IEC 14165 series, under the general title *Information technology - Fibre channel,* can be found on the IEC web site.

This International Standard has been approved by vote of the member bodies and the voting results may be obtained from the address given on the second title page.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

INTRODUCTION

This International Standard defines an application programming interface (API) by which a storage application may perform the functions of one or more SCSI Targets or Initiators, and control high-performance command/data forwarding and manipulation facilities.

INFORMATION TECHNOLOGY – FIBRE CHANNEL –

Part 521: Fabric application interface standard (FAIS)

1 Scope

This part of ISO/IEC 14165 describes a set of functions and data structures in the C language abstracting the details of the FAIS_Platform from the implementation of a storage management application.

This standard defines an API only in the C language. Functionally equivalent APIs may be implemented in other languages but these are beyond the scope of this part of ISO/IEC 14165. All functions provided to operate with function specifications defined in this standard shall use C-style calling conventions. This constraint does not limit the internal implementation of components of a FAIS_Provider.

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

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, this is not required by this standard.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document, including any amendments, applies.

The provisions of the referenced specifications other than ISO/IEC, IEC, ISO and ITU documents, as identified in this clause, are valid within the context of this International Standard. The reference to such a specification within this International Standard does not give it any further status within ISO or IEC. In particular, it does not give the referenced specification the status of an International Standard.

ISO/IEC 9899:1999, *Programming languages – C*

ISO/IEC 10646:2003, *Information technology - Universal Multiple-Octet Coded Character Set (UCS)*

ANSI INCITS 408-2005, *Information technology – Fibre Channel – Framing and Signaling-2 (FC-FS-2)*

ISO/IEC 14776-453, *Information technology – Small computer system interface (SCSI) – Part* 453: *SCSI Primary Commands-3 (SPC-3)* [ANSI INCITS 408-2005]

ISO/IEC 19501: *Information technology – Open distributed processing – Unified modeling language (UML), Version 1.4.2*

NOTE For more information on UML specifications, contact the Object Modeling Group at http://www.omg.org.

INCITS Project 1828-D: *Fibre Channel Protocol for SCSI, Fourth Version (FCP-4)*

RFC 791, *Internet Protocol*

RFC 2279, *UTF-8, a transformation format of ISO 10646*

RFC 2460, *Internet Protocol, Version 6 (IPv6)*

RFC 3720, *Internet Small Computer Systems Interface (iSCSI)*

3 Definitions and conventions

3.1 Overview

For this standard, the following definitions, conventions, abbreviations, acronyms, and symbols apply.

3.2 Terms and definitions

3.2.1
Activate
In the context of a FAIS_SCSI_Port, the process of making an FT or BI accessible to the SCSI transport. A FAIS_SCSI_Port needs to be activated before it is visible to other nodes in the storage network.

3.2.2
application programming interface (API)
interface that provides the means for higher-level software (i.e., applications) to control a specialized subsystem

3.2.3
asynchronous function call
function call that returns when the request is scheduled to be performed. When all of the processing for the request is completed, a subsequent callback indicating the completion status is generated

3.2.4
Back-end
A qualifier for FAIS constructs that issue SCSI commands

3.2.5
BI
back-end Initiator FAIS_SCSI_Port

3.2.6
BIT
back-end I_T nexus

3.2.7
BITL
A back-end I_T_L nexus

3.2.8
BITLSet
group of BITLs

3.2.9
Concatenation
logical joining of two or more contiguous ranges of logical blocks

3.2.10
Control Path Processor (CPP)
computational entity in which the FAIS_Provider processes FAIS_Client invocations of the FAIS_API

3.2.11
Data Path Controller (DPC)
computational entity to which the FAIS_Provider delegates some computations (e.g., SCSI processing)

3.2.12
Drain
Wait until there are no offloaded I/Os that are actively being processed by the FAIS_Platform. An I/O is no longer considered to be actively processed when it is either fully completed or its processing has been halted for any reason (e.g., when an offloaded I/O is held by a Quiesce operation or faulted to the FAIS_Client).

3.2.13
FAIS_API
API defined in this standard

3.2.14
FAIS_Client
instance of an application that runs on the CPP. The entity that invokes the FAIS_API

3.2.15
FAIS_Platform
hardware portion of a FAIS-based storage subsystem, that generally consists of two primary components, the Control Path Processor (CPP) and the Data Path Controller (DPC)

3.2.16
FAIS_Portal
addressable endpoint in the storage network

3.2.17
FAIS_Provider
instance of the provider of the FAIS_API

3.2.18
FAIS_Region
abstraction representing a subset of the resources provided by a FAIS_Platform for exposing FAIS_SCSI_Ports

3.2.19
FAIS_SCSI_Port
SCSI Port used by the FAIS_Provider (e.g., in Fibre Channel an Nx_Port supporting FCP). Either an Initiator (BI) or Target (FT)

3.2.20
Fault
operation performed by the FAIS_Provider that halts the processing of an offloaded I/O and generates an event notification to the FAIS_Client. The operation occurs when the FAIS_Provider encounters a condition that requires assistance from the FAIS_Client, that may occur during any phase of the I/O. The I/O remains offloaded but its processing does not continue until it is explicitly allowed by the FAIS_Client.

3.2.21
FIT
Front-end I_T nexus

3.2.22
FITL
Front-end I_T_L nexus

3.2.23
Forward
operation performed by the FAIS_Provider that transfers the processing responsibility for an I/O to the FAIS_Client. It is the responsibility of the FAIS_Client to handle all phases of a forwarded I/O.

3.2.24
Front-end
qualifier for FAIS constructs that receive SCSI commands

3.2.25
FT
Front-end Target FAIS_SCSI_Port

3.2.26
I_T_L Nexus
relationship between two SCSI devices defined in SAM-3. Specifically, it is the relationship between a SCSI initiator port, a SCSI target port, and a logical unit

3.2.27
Logical Block
set of data bytes accessed and referenced as a unit

3.2.28
Logical Block Address (LBA)
value used to reference a logical block.

3.2.29
Logical Unit (LU)
as defined in SAM-3, a SCSI target device object, containing a device server and task manager, that implements a device model and manages tasks to process SCSI commands sent by an application client

3.2.30
Mappings
metadata that is provided by the FAIS_Client to allow the FAIS_Provider to perform I/O processing. The metadata contains conversions between two data address spaces (e.g., physical disk block addresses and the block addresses of the virtual disks presented to operating environments by control software).

3.2.31
Metadata
data that describes other data. In disk arrays, metadata consists of items such as array membership, member extent sizes and locations, descriptions of logical disks and partitions, and array state information. In file systems, metadata includes file names, file properties and security information, and lists of block addresses where each file's data is stored. In this standard, metadata includes the Mappings.

3.2.32
Mirroring
mapping technique in which ranges of logical blocks of a virtual disk are mapped to two or more identical copies of data

3.2.33
network byte order
order in which the bytes of a multi-byte value are transmitted on a network, most significant byte first, next most significant byte following, and so on. For values in memory, the most significant byte is stored at the memory location with the lowest address, the next byte in significance, is stored at the next higher memory location, and so on.

3.2.34
Offload
transfer of processing responsibility from the FAIS_Client to the FAIS_Provider

3.2.35
operation
unit of data processing supported by the FAIS_API

3.2.36
Quiesce
operation performed by the FAIS_Client that establishes a state on a FAIS Object (e.g., on a VDEV), such that new offloaded I/Os arriving after this point are held by the FAIS_Platform and drains existing I/Os. The processing of these held I/Os does not begin until a Resume operation is performed by the FAIS_Client

3.2.37
Remote SCSI Port
SCSI Port that may be used in an I_T nexus with a FAIS_SCSI_Port. A SCSI Port is synonomous with the service delivery port. The service delivery port is defined in SAM-3

3.2.38
Resume
operation performed by the FAIS_Client that establishes a state on a FAIS Object (e.g., on a VDEV), such that the processing of any new offloaded I/Os, including those previously held by a Quiesce operation, may begin immediately

3.2.39
SCSI Port
see SAM-3

3.2.40
Striping
mapping technique in which consecutive ranges of logical block addresses of a virtual disk are mapped to successive disk members in a cyclic pattern

3.2.41
synchronous function call
function call that returns only when all requested processing is completed

3.2.42
Synchronous Mirroring
type of mirroring implementation where each write operation is not acknowledged as completed until the write operations to all identical copies are completed

3.2.43
Task Management Function
operations defined in SAM-3 for task management

3.2.44
Transport Protocol
protocol utilized by the FAIS_Portal. For FAIS, this includes FCP in a FibreChannel-based storage network and iSCSI in an IP-based storage network.

3.2.45
UTF-8
character set that is a transformation format of the character set defined by ISO 10646 (see RFC 2279).

3.2.46
VDEV
class in the FAIS object model used to represent a SCSI addressable set of consecutively numbered logical blocks with an associated organization

3.3 Editorial Conventions

In this standard, a number of conditions, mechanisms, sequences, parameters, events, states, or similar terms are printed with the first letter of each word in uppercase and the rest lowercase (e.g., Exchange, Class). Any lowercase uses of these words have the normal technical English meanings.

Lists sequenced by letters (e.g., a-red, b-blue, c-green) show no priority relationship between the listed items. Numbered lists (e.g., 1-red, 2-blue, 3-green) show a priority ordering between the listed items.

The ISO convention of numbering is used (i.e., the thousands and higher multiples are separated by a space and a comma is used as the decimal point.) A comparison of the American and ISO conventions is shown in table 1.

Table 1 – ISO and American Conventions

ISO	American
0,6	0.6
1 000	1,000
1 323 462,9	1,323,462.9

In case of any conflict between figure, table, and text, the text, then tables, and finally figures take precedence.

In all of the figures, tables, and text of this document, the most significant bit of a binary quantity is shown on the left side.

When the value of a bit or field is not relevant, x or xx appears in place of a specific value.

Unless stated otherwise, 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, and numbers or upper case letters immediately followed by lower-case h (xxh) are hexadecimal values.

A numeric range is indicated by listing the two extremes separated by ".." (e.g., "1 .. 6" indicates the range from 1 to 6, including 1 and 6).

3.4 Abbreviations and acronyms

Abbreviations and acronyms applicable to this standard are listed. Definitions of several of these items are included in 3.2.

< Less than
>= Greater than or equal to
API Application Programming Interface
BI Back-end Initiator
BIT Back-end Initiator-Target Nexus
BITL Back-end Initiator-Target-LUN Nexus
CDB SCSI Command Descriptor Block (see SAM-3)
CPP Control Path Processor
DPC Data Path Controller
FCP Fibre Channel Protocol (see FCP-4)
FIT Front-end Initiator-Target Nexus
FITL Front-end Initiator-Target-LUN Nexus
FLUVDEV Front-end Logical Unit
FT Front-end Target
I/O Input/Output
ISID Initiator Session ID (see RFC 3720)
IP Internet Protocol
IPv4 Internet Protocol version 4 (see RFC 791)

IPv6 Internet Protocol version 6 (see RFC 2460)
iSCSI Internet Small Computer Systems Interface (see RFC 3720)
LBA Logical Block Address
LU Logical Unit
LUN Logical Unit Number (see SAM-3)
RAID Redudant Array of Indpendent Disks
SCB SCSI Control Block
SCSI Small Computer System Interface
TCP Transmission Control Protocol
TPGT Target Portal Group Tag (see RFC 3720)
UTF-8 Universal Transformation Format 8
VDEV Virtual Device
WWN Worldwide_Name

3.5 Notation for Procedures and Functions

Procedures and functions are specified in the syntax of the C programming language (see ISO/IEC 9899).

3.6 Enumeration Lists

In this standard constants may be defined using the C programming language enum construct (see ISO/IEC 9899). For any instances where the enum construct is used to define constants, values not defined shall be reserved unless stated otherwise.

3.7 Class-related definitions

3.7.1
aggregation
form of association that defines a whole-part relationship between the whole (i.e., the aggregate) and its parts

3.7.2
association
relationship between two or more classes that specifies that objects of one class are connected to objects of another class

3.7.3
attribute
named property of a class that describes the range of values that the class or its objects may hold

3.7.4
constraint
mechanism for specifying semantics or conditions that are maintained as true between entities (e.g., a required condition between associations)

3.7.5
class
description of a set of objects that share the same attributes, operations, relationships, and semantics. Classes may have attributes and may support operations

3.7.6
dependency
relationship between two classes where a change to one class (i.e., the independent class) may cause a change in the other class (i.e., the dependent class)

3.7.7
generalization
relationship among classes where one class (i.e., the superclass) shares the attributes and/or operations on one or more classes (i.e., the subclasses)

3.7.8
multiplicity
indication of the range of allowable instances that a class or an attribute may have

3.7.9
object
entity with a well-defined boundary and identity that encapsulates state and behavior. All objects are instances of classes (i.e., a concrete manifestation of a class is an object)

3.7.10
operation
service that may be requested from any object of the class in order to effect behavior. Operations describe what a class is allowed to do and may be a request or a question. A request may change the state of the object but a question should not.

3.7.11
role
label at the end of an association or aggregation that defines a relationship to the class on the other side of the association or aggregation

3.8 Class diagram conventions

The notation in this subclause is based on the Unified Modeling Language (UML) specification.

Figure 1 shows the notation used for classes in class diagrams.

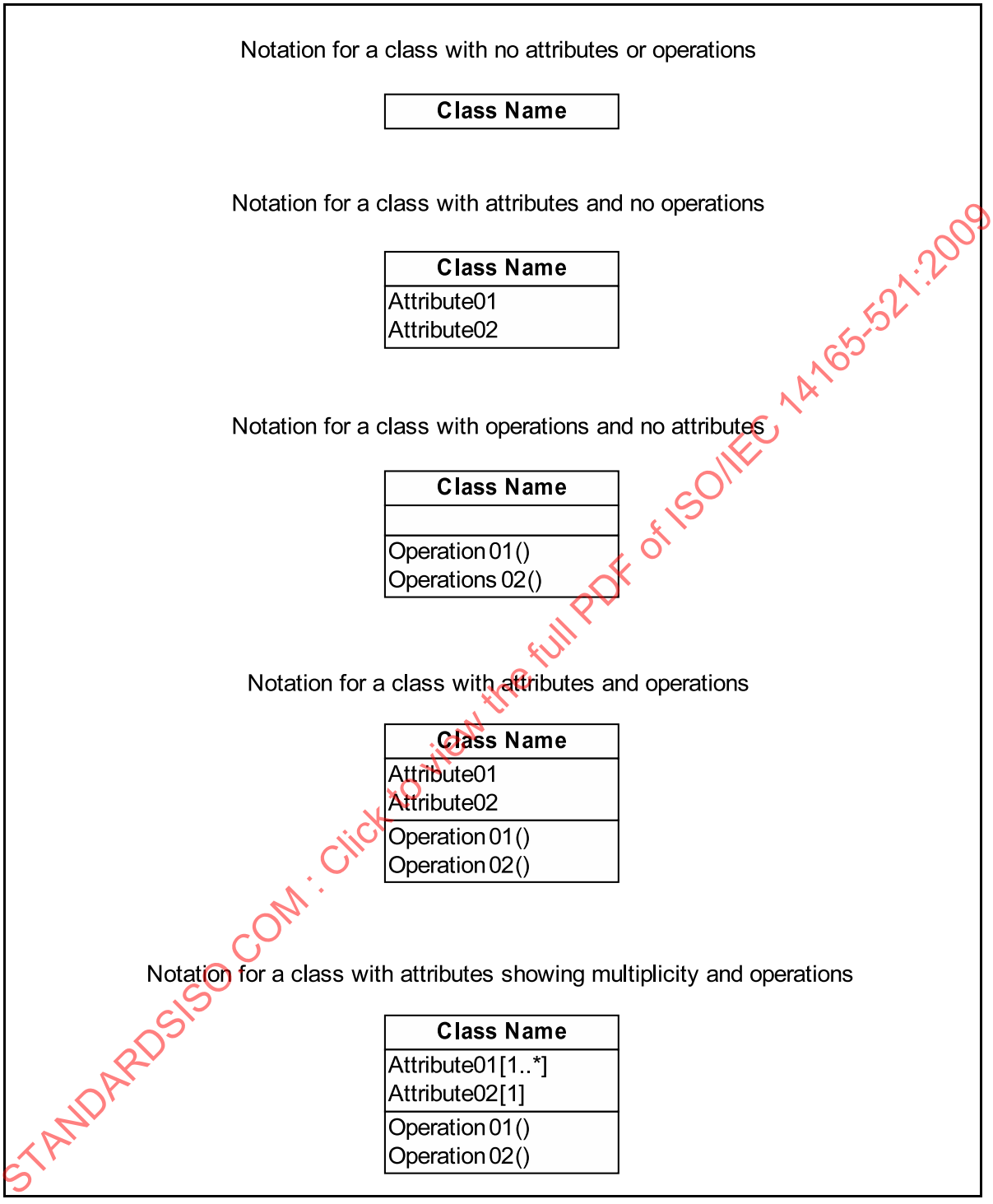

Figure 1 – Class diagram conventions

See table 2 for the notation used to indicate multiplicity for objects and attributes.

Table 2 – Multiplicity notation

Object Notation	Description
	The number of instances of an attribute is not specified.
1	One instance of the object or attribute exists.
0..*	Zero or more instances of the object or attribute exist.
1..*	One or more instances of the object or attribute exist.
0..1	Zero or one instance of the object or attribute exists.
n..m	n to m instances of the object or attribute exists (e.g., 2..8).
x, n..m	Multiple disjoint instances of the object or attribute exists (e.g., 2, 8..15).

For the notation describing the relationships that may exist between classes, see:

a) Figure 2 for association;

b) Figure 3 for aggregation; and

c) Figure 4 for generalization.

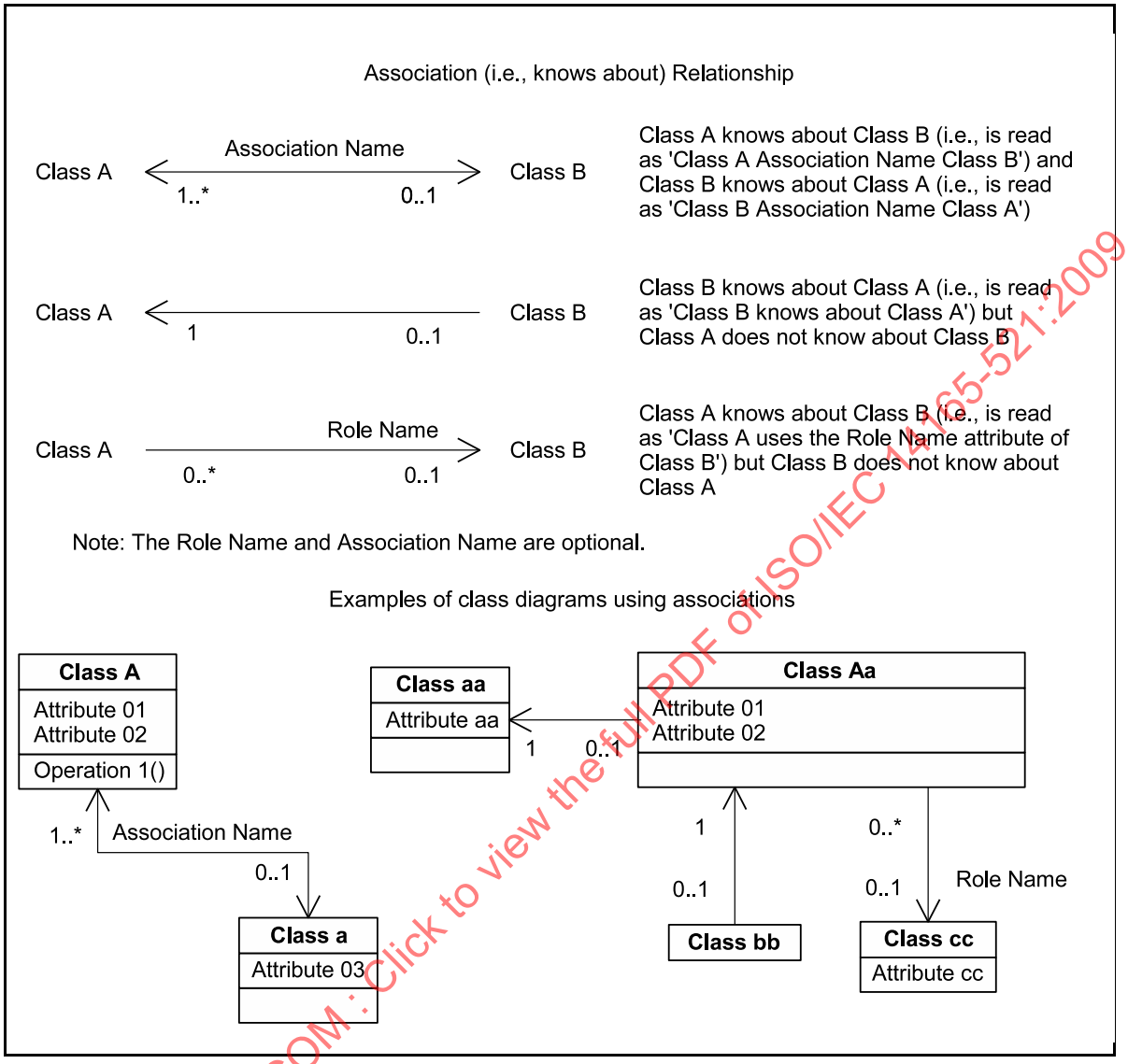

Figure 2 – Notation for association relationships for class diagrams

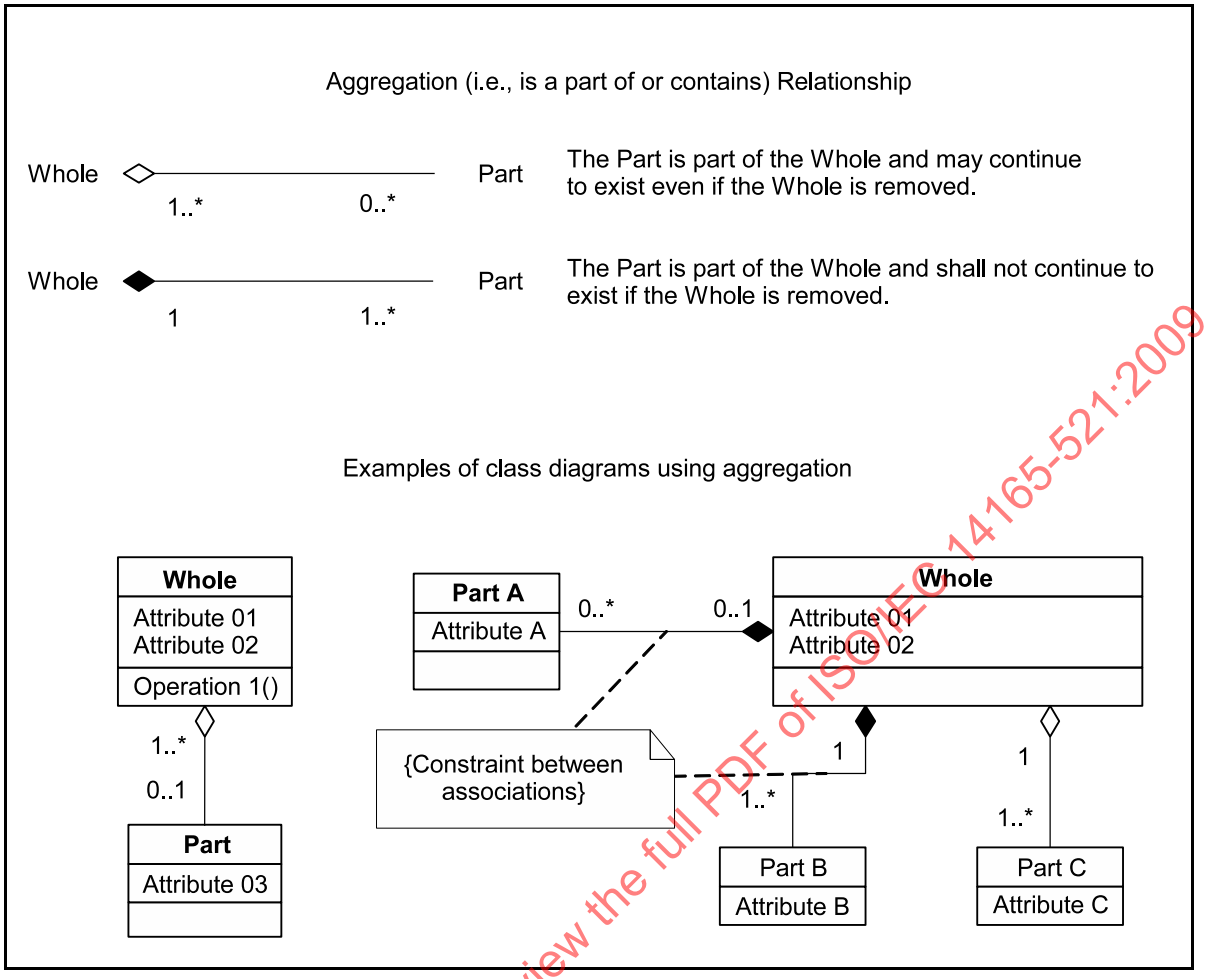

Figure 3 – Notation for aggregation relationships for class diagrams

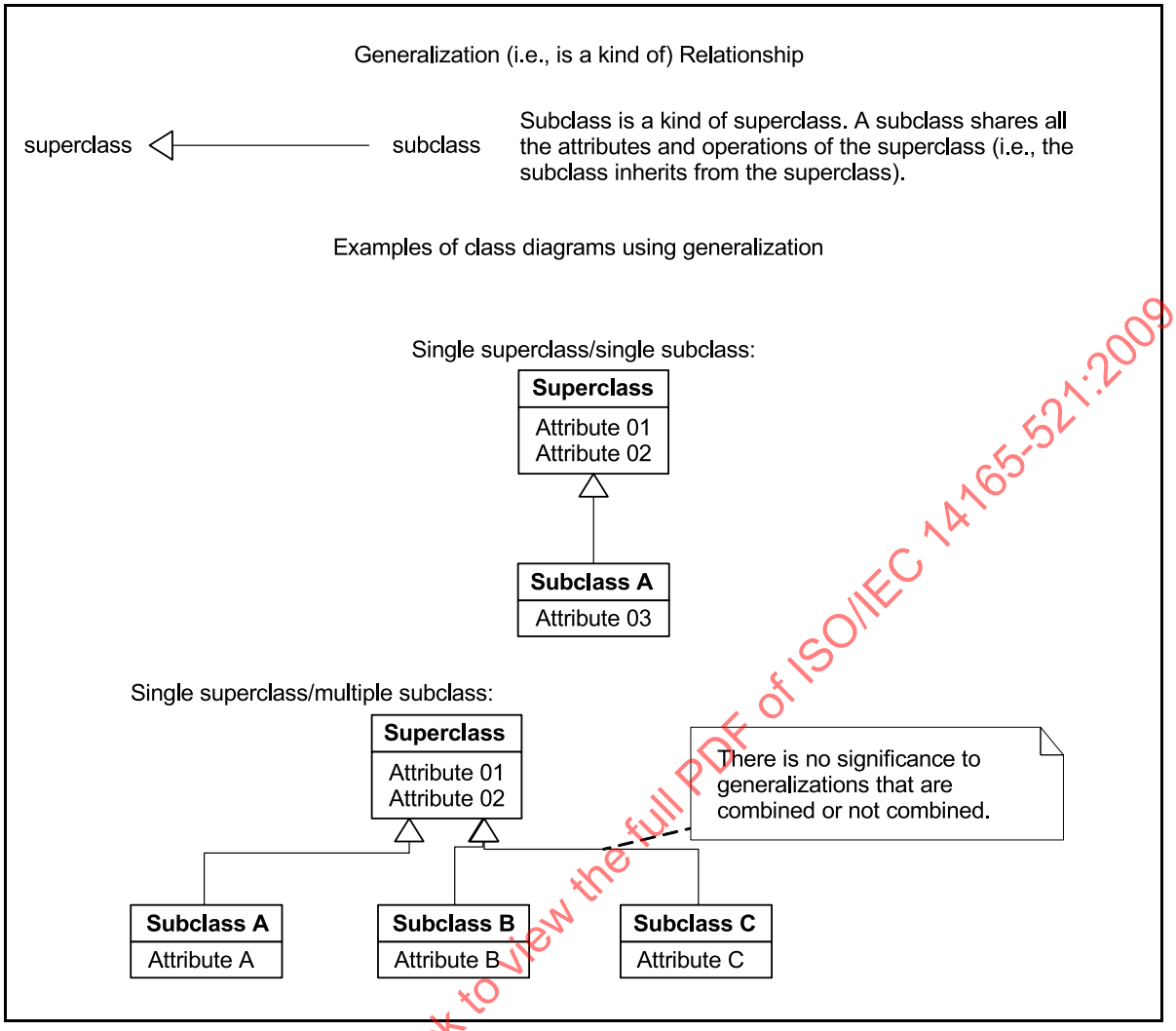

Figure 4 – Notation for generalization relationships for class diagrams

3.9 Keywords

3.9.1
expected
keyword used to describe the behavior of the hardware or software in the design models assumed by this standard. Other hardware and software design models may also be implemented

3.9.2
ignored
keyword used to describe an unused bit, byte, word, field or code value. The contents or value of an ignored bit, byte, word, field or code value shall not be examined by the receiving device and may be set to any value by the transmitting device

3.9.3
invalid
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

3.9.4
mandatory
keyword indicating an item that is required to be implemented as defined in this standard

3.9.5
may
keyword that indicates flexibility of choice with no implied preference (i.e., equivalent to may or may not)

3.9.6
may not
keywords that indicates flexibility of choice with no implied preference (i.e., equivalent to may or may not)

3.9.7
obsolete
keyword indicating that an item was defined in prior standards but has been removed from this standard

3.9.8
opaque
keyword indicating that value has no semantics or internal structure with respect to the specfied entity

3.9.9
optional
keyword that describes features that are not required to be implemented by this standard. However, if any optional feature defined by this standards is implemented, it shall be implemented as defined in this standard

3.9.10
reserved
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 future versions of 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.

3.9.11
shall
keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products that conform to this standard

3.9.12
should
keyword indicating flexibility of choice with a strongly preferred alternative; equivalent to the phrase "it is strongly recommended"

3.9.13
x or xx
value of the bit or field is not relevant

3.10 T10 Vendor ID fields

A T10 Vendor ID shall be a string of one to eight characters that is recorded in an informal list of Vendor IDs maintained by INCITS Technical Committee T10 (see http://www.t10.org).

A field described as containing a T10 Vendor ID shall contain the first character of the T10 Vendor ID in the most significant byte of the field, and successive characters of the T10 Vendor ID in successively less significant bytes of the field. Any bytes of the field not filled by characters of the T10 Vendor ID shall be filled with ASCII space characters (20h).

4 Operational model

4.1 Overview

FAIS supports storage functions resident on an equipment within a storage network infrastructure (e.g., a Fibre Channel Fabric). FAIS defines an application programming interface (API) and associated data structures. The API defines an abstraction that obviates the need for a FAIS_Client to be concerned with the implementation detail of a FAIS_Platform.

FAIS supports storage functions that perform classic enterprise-level storage transformation processes (e.g., virtualization and RAID). Virtualization is an abstraction process by which the boundaries of the physical storage devices are hidden from operating systems (e.g., eight 80 gigabyte disks may appear to an operating system as a single 640 gigabyte virtual disk). RAID (i.e., Redundant Arrays of Independent Disks) is a process that uses a set of physical storage devices to hold redundancy information as well as data so that the failure of a physical storage device does not result in any visible data loss.

FAIS does not define the command sets used to control the storage devices, but instead leverages the existing SCSI command sets (e.g., SPC-3).

This standard defines an API by which a storage application may:

a) perform all of the functions of one or more SCSI Targets (see SAM-3);

b) perform all of the functions of one or more SCSI Initiators (see SAM-3);

c) configure and control high-performance command/data forwarding and manipulation facilities present in the underlying equipment; and

d) delegate the processing of specific SCSI command types addressed to specific entities of those facilities.

This standard incorporates models that demonstrate how the FAIS_API may be applied to implementing such storage applications in a storage networking environment. Such models describe the functions that a storage application may use, including:

a) identification and initialization of real and virtual storage resources;

b) establishment of parameters and mappings that control SCSI storage operation;

c) processing of supporting SCSI commands on behalf of a virtual storage device; and

d) processing of data distribution functions, data replication functions, and data journaling functions appropriate to a storage application.

The FAIS_API defined in this standard describes an interface used by FAIS_Clients to access the services of a FAIS_Platform.

The following general requirements are to be met. The FAIS_API:

a) is Operating System independent;

b) supports aborting Asynchronous calls;

c) supports setting timeout for each call; and

d) supports re-entrant and thread-safe coding.

4.2 Operational layering

Figure 5 shows the abstract model of the operational layering.

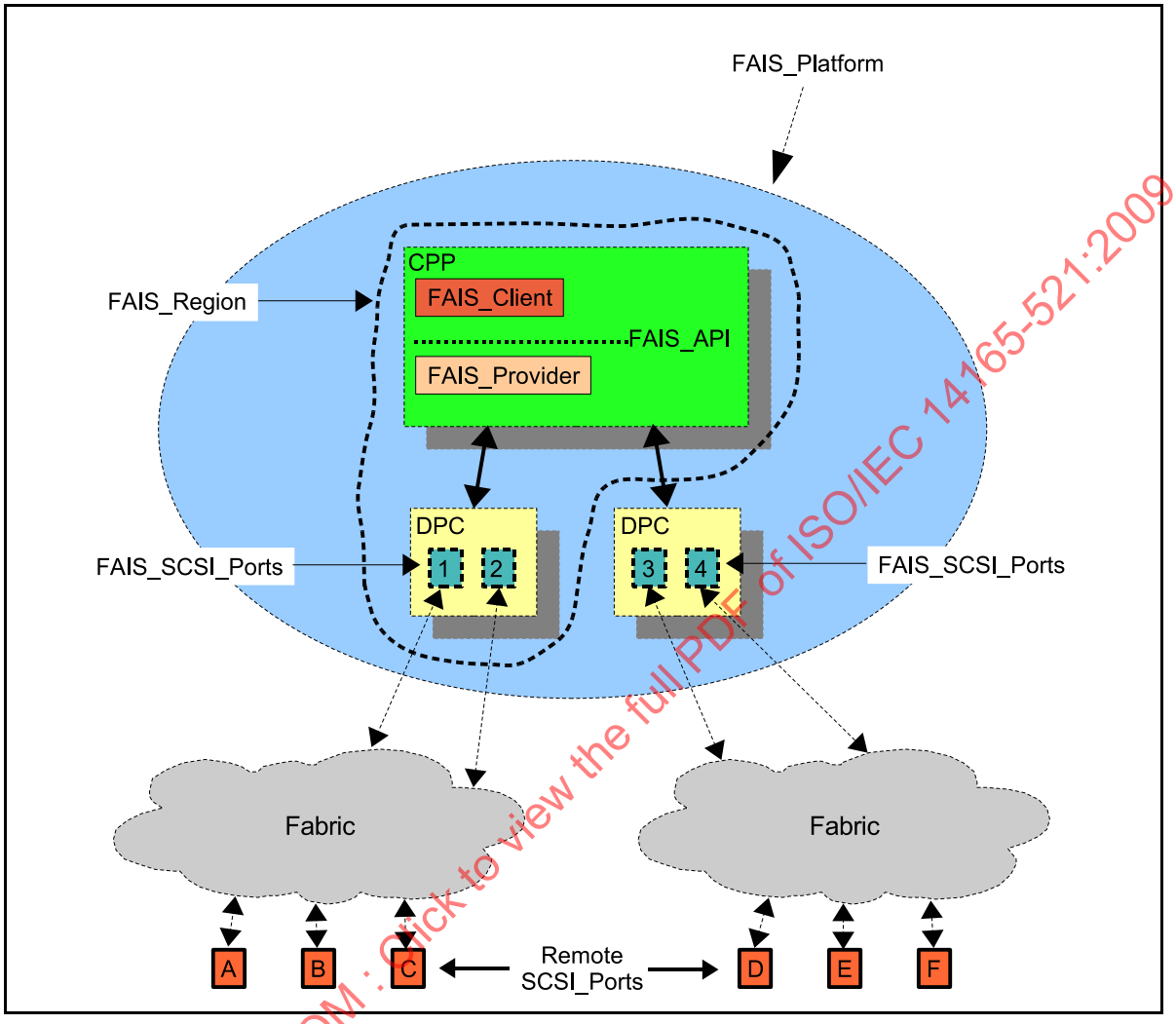

Figure 5 – Operational layering

The FAIS_Client is the portion of a storage application that communicates with the FAIS_Provider using the FAIS_API. The storage application is expected to perform storage functions such as virtualization and RAID. The FAIS_Client is hosted by a Control Path Processor (CPP).

The CPP hosts the storage application (i.e., FAIS Clients) whose functionality may include initialization, data path configuration, exception handling, management operations and the maintenance of other state information. FAIS_Clients invoke FAIS_Provider functions using API Calls defined by this standard. FAIS_Clients are responsible for building structures and mappings that reflect the intended data layout (i.e., abstraction) and accessibility. FAIS_Clients may provide the control logic for data movement functions (e.g., snapshot copy, on-line migration, remote mirroring, asynchronous replication). FAIS_Clients are Fabric-based but not necessarily switch-based. The FAIS architecture is intended to scale the number of FAIS_Clients to provide distributed or clustered functionality across the Fabric.

FAIS_Providers process API calls from FAIS_Clients. The FAIS_Provider translates information received from the FAIS_Client into an appropriate form for use by a Data Path Controller and vice versa. A FAIS_Provider is hosted by a CPP and may manage one or more DPCs.

For the purposes of this standard, a Data Path Controller is a high performance frame processing device. The DPC is programmed (i.e., controlled) by a control path. Once the initial configuration is programmed, a DPC may process I/Os without intervention from a control path.

The FAIS_SCSI_Ports are SCSI Ports (see SAM-3) used by a FAIS_Provider. As defined in this standard, FAIS_SCSI_Ports are abstractions that function as SCSI Ports within the FAIS_Platform and through which one or more DPCs receive and transmit information using the applicable SCSI command set standards and SCSI transport protocol standards (see SAM-3).

The remote SCSI_Ports are SCSI Ports (see SAM-3) that may be used in an I_T nexus with a FAIS_SCSI_Port. As defined in this standard, remote SCSI_Ports are SCSI Ports outside of the FAIS_Platform and communicate with the FAIS_SCSI_Ports.

4.3 Client/Provider model

The model is based on the following design considerations:

a) provide a generic, implementation-independent description of a volume;

b) decouple the FAIS_Client implementation from the internal data structures used by the DPC(s);

c) provide methods to support high level operations (e.g., snapshot copy, online data migration) that are beyond the scope of this standard;

d) require that persistent metadata updates related to configuration are handled by the FAIS_Client; and

e) require that storage allocation is handled by the FAIS_Client.

4.4 Service groups

4.4.1 Overview

The FAIS_API is organized into the following major service groups.

4.4.2 General services

General Services include those types, data structures or functions that may be used by the other defined services.

4.4.3 Port services

Port Services provide support for creating, removing and performing operations on SCSI Ports.

4.4.4 Front-end services

Front-end Services provide support for processing requests that are received from a remote SCSI Initiator.

4.4.5 Back-end services

Back-end Services provide support for discovering elements connected to the logical back-end of the FAIS_Platform and for sending commands to devices on the back-end.

4.4.6 Volume management services

Volume Management Services provide support for building a mapping between storage entities presented to storage consumers on the front-end of the FAIS_Platform and storage entities accessed by BITLSets on the back-end. Volume Management Services also provide support for managing creation and updates of VDEVs.

4.5 Framework

4.5.1 Function call type

Each FAIS_API operation is defined with either a synchronous-only or an asynchronous-capable request method, not both.

A synchronous-only request method has the following characteristics:

a) supports only synchronous function calls;

b) all parameters are in the function call argument list (i.e., no function parameter block, see 4.5.3);

c) completion is upon function call return; and

d) completion status is in the function call return value.

An asynchronous-capable request method has the following characteristics:

a) supports both synchronous and asynchronous function calls, depending on whether a callback function is specified in the function parameter block (i.e., non-NULL);

b) all parameters are in a caller-allocated data structure associated with each request (i.e., the function parameter block, see 4.5.3);

c) completion is upon function call return if synchronous or callback function if asynchronous; and

d) completion status is in the function call return value if synchronous or callback function argument list if asynchronous.

4.5.2 Requests and completions

A FAIS_API operation is performed synchronously either when a synchronous-only request method is called, or an asynchronous-capable request method is called with a NULL callback function. In all synchronous function calls, the operation completes upon return from the function call and the return value of the call indicates the completion status.

A FAIS_API operation is performed asynchronously when an asynchronous-capable request method is called with a non-NULL callback function. In all asynchronous function calls, the return value of the function call indicates whether the request was successfully accepted by the callee. If the return value is FAIS_STATUS_SUCCESS, the request was successfully accepted and the callback function shall be called when the operation completes; in this case, the completion status is provided in the callback function's argument list. If the return value is not FAIS_STATUS_SUCCESS, the request completed unsuccessfully (i.e., the callback function is not called), and the completion status is indicated by the return value.

FAIS_API operations are thread-safe and reentrant (e.g., multiple requests of the same operation may be outstanding concurrently).

4.5.3 Function parameter block

Each asynchronous-capable request method accepts a pointer to a data structure that specifies its input and output arguments called the function parameter block. The caller shall be responsible for allocating the memory for the function parameter block for each request, and this memory shall remain valid for the duration of the request (i.e., it may not be freed/reused until the operation completes either synchronously via the function call return or asynchronously via the callback function call).

When an asynchronous-capable request method is called, the caller shall initialize all fields designated as input parameters in the function parameter block prior to submitting the request and it may not sub-

sequently modify any fields in the function parameter block of a successfully submitted request until the operation completes.

Upon completion of the request, the completion status indicates whether the remaining output fields in the function parameter block are valid. If the status field indicates a successful completion, all output parameters in the function parameter block shall be valid.

In general, the entity (i.e., FAIS_Client or FAIS_Provider) that allocates memory resources shall also deallocate (i.e., free) them when appropriate. In the case of the function parameter block, the caller is responsible for freeing any memory allocated for the function parameter block after the operation completes.

The address of the function parameter block submitted in each request uniquely identifies that request. The function parameter block address may be used by the caller to request cancellation of the operation. This standard supports the cancellation of requests successfully accepted by asynchronous-capable request methods. It is implementation-dependent as to whether a particular request may be cancelled.

4.6 Event notification

Apart from the FAIS_Client making requests to the FAIS_Provider and receiving asynchronous responses, there are also cases where the FAIS_Provider requests a service from or delivers an event to the FAIS_Client. This is an asynchronous, unsolicited event.

NOTE Event Notification is to be addressed in a future revision of this standard.

5 Object model

5.1 Overview

The object model provides to developers of FAIS_Clients the abstractions necessary to control the processing of I/Os handled by a FAIS_Provider. The FAIS Object Model is intended to be independent from any specific implementation of a FAIS_Provider.

The object model describes only objects needed to both FAIS_Client and FAIS_Provider and does not describe private objects created by them in support of their implementation.

5.2 Model

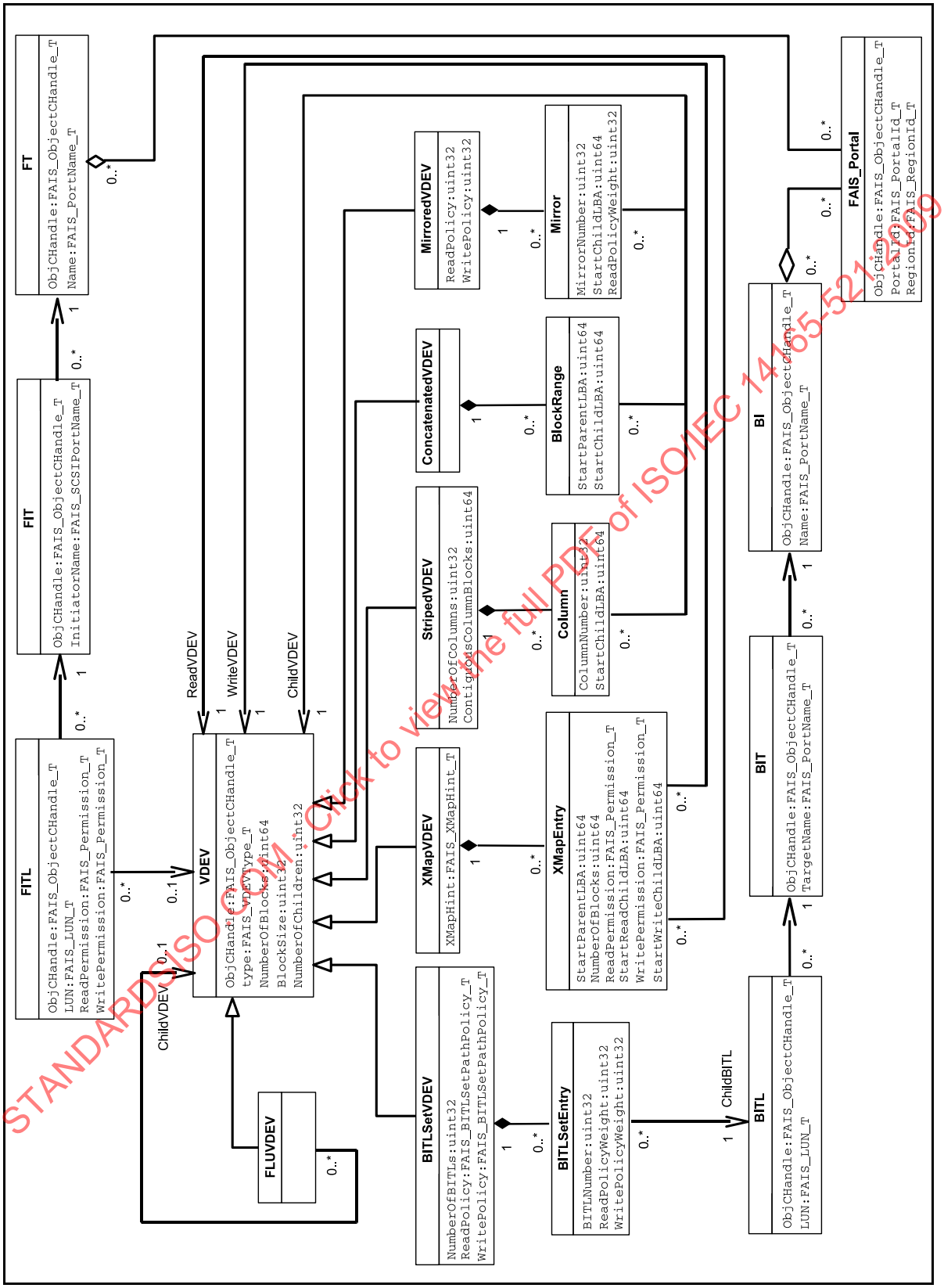

Figure 6 – Object model

5.3 Meta-Attributes

5.3.1 Description

Short description of the object's purpose within the model, including whether the object is created implicitly by the FAIS_Provider or created explicitly by the FAIS_Client.

5.3.2 Attributes

List and definitions of the objects attributes. Each attribute is also described as:

a) static: set by the object creator;

b) dynamic: changed dynamically by the provider in response to events; or

c) settable: changed explicitly by the client.

5.3.3 Relationships

The list of relationships with other objects within the model (e.g., inheritance type relationships, cardinality).

5.3.4 Handles

5.3.4.1 FAIS_ClientCHandle

Use Case: When a FAIS_Provider delivers a callback for an asynchronous function call to the FAIS_Client, FAIS_ClientCHandle is provided by the FAIS_Provider to identify the FAIS_Client instance in the FAIS_Client space.

Each FAIS_Client that registers with the FAIS_Provider through fais_Init() shall be assigned a FAIS_ClientCHandle assigned by the FAIS_Client.

The FAIS_ClientCHandle has the following characteristics:

a) the FAIS_ClientCHandle shall be persistent between the fais_Init() and the fais_DeInit() function calls called by the FAIS_Client;

b) the FAIS_ClientCHandle is opaque to the FAIS_Provider; and

c) the FAIS_ClientCHandle, provided at the time of FAIS_Client registration, shall be specified by the FAIS_Provider in all the subsequent callbacks and events.

5.3.4.2 FAIS_ClientPHandle

Use Case: When a FAIS_Client issues a FAIS_API function call, FAIS_ClientPHandle is provided by the FAIS_Client to identify the FAIS_Client instance in the FAIS_Provider space.

Each FAIS_Client that registers with the FAIS_Provider through fais_Init() shall be assigned a FAIS_ClientPHandle assigned by the FAIS_Provider.

The FAIS_ClientPHandle has the following characteristics:

a) the FAIS_ClientPHandle shall be unique to the FAIS_Client among all the Clients interfacing with the same FAIS_Provider. The new FAIS_ClientPHandle for the FAIS_Client shall not conflict with the other FAIS_Client FAIS_ClientPHandle's assigned by the FAIS_Provider;

b) the FAIS_ClientPHandle shall be persistent between the fais_Init() and the fais_DeInit() function calls called by the FAIS_Client;

c) the FAIS_ClientPHandle may be reused after the FAIS_Client has unregistered with the FAIS_Provider through fais_DeInit();

d) the FAIS_ClientPHandle is opaque to the FAIS_Client; and

e) the FAIS_ClientPHandle, provided at the time of FAIS_Client registration, shall be specified by the FAIS_Client in all the subsequent FAIS_API calls.

5.3.4.3 FAIS_ObjectCHandle

Use Case: When a FAIS_Provider delivers a callback for an asynchronous function call to the FAIS_Client, FAIS_ObjectCHandle is provided by the FAIS_Provider to identify the FAIS object instance in the FAIS_Client space.

Each FAIS object (e.g., FITL) that may be created and managed shall be assigned a FAIS_ObjectCHandle assigned by the FAIS_Client)

The FAIS_ObjectCHandle of a FAIS object has the following characteristics:

a) the FAIS_ObjectCHandle shall be persistent between creation and subsequent destruction of the FAIS object; and

b) when a FAIS_Provider references a FAIS object (e.g. in a callback), it shall provide the FAIS_ObjectCHandle specified by the FAIS_Client.

The FaisObjectCHandle is assigned by the FAIS_Client. The FAIS_Provider shall not implicitly modify the FaisObjectCHandle.

5.3.4.4 FAIS_ObjectPHandle

Use Case: When a FAIS_Client issues a FAIS_API function call, FAIS_ObjectPHandle is provided by the FAIS_Client to identify the FAIS object instance in the FAIS_Provider space.

Each FAIS object (e.g., FITL) that may be created and managed shall be assigned a FAIS_ObjectPHandle assigned by the FAIS_Provider.

The FAIS_ObjectPHandle of a FAIS object has the following characteristics:

a) the FAIS_ObjectPHandle of a FAIS object shall remain valid until destruction of the FAIS object or deinitialization of the FAIS_Client;

b) if a FAIS object is to be passed in a FAIS_API call, FAIS_ObjectPHandle shall be used by the FAIS_Client to reference a FAIS object;

c) a FAIS object may persist after the deinitialization of the FAIS_Client; and

d) the FAIS_ObjectPHandle is valid only within the context of a valid FAIS_ClientPHandle. A FAIS object may be identified by a different FAIS_ObjectPHandle within the context of a different FAIS_ClientPHandle.

When a FAIS object is modified, the FAIS_ObjectPHandle assigned to the FAIS object shall not change. After modification, the FAIS object is still uniquely identified by the same FAIS_ObjectPHandle.

5.3.5 Identifiers

5.3.5.1 FAIS_Portal_id

Each FAIS_Portal (i.e., Initiator, Target, or both) is identified by an address defined in the SCSI transport protocol (e.g., iSCSI, FCP) specification. In FAIS, the FAIS_Portal address is stored in the FAIS_Portal_id.

The FAIS_Portal_id of a FAIS_Portal has the following characteristics:

a) the FAIS_Portal_id uniquely identifies a FAIS_Portal among all the other FAIS_Portals; and

b) the FAIS_Portal_id is compliant with the transport protocol specification.

A FAIS_Client shall know the Transport protocols used by the FAIS_Provider. The FAIS_Client may generate a FAIS_Portal_id for the FAIS_Portal it creates and manages.

5.3.5.2 FAIS_PortName

Each SCSI Port (i.e., Initiator or Target) is identified by a name defined by the SCSI Transport protocol (e.g., iSCSI, FCP) specification. In FAIS, this name is stored in the FAIS_PortName.

The FAIS_PortName of a FAIS_SCSI_Port shall have the following characteristics:

a) the FAIS_PortName uniquely identifies a FAIS_SCSI_Port among all the SCSI Ports it may communicate with;

b) the FAIS_PortName is compliant with the transport protocol specification; and

c) a FAIS_PortName may be shared between a BI and an FT.

A FAIS_Client shall know the Transport protocols used by the FAIS_Provider. The FAIS_Client may generate the FAIS_PortName for the FAIS_SCSI_Port it creates and manages, or may allow the FAIS_Provider to generate the FAIS_PortName.

5.3.5.3 FAIS_RegionId

Each SCSI port (i.e., Target, Initiator, or both) is potentially associated with a subset of a FAIS_Platform's resources. In FAIS, each subset resource group is called a FAIS_Region and is uniquely identified by the contents of the structure FAIS_RegionId.

The FAIS_RegionId of a FAIS_Region shall have the following characteristics:

a) the FAIS_RegionId identifies a subset of resources of a FAIS_Platform;

b) the FAIS_RegionId is unique within a FAIS_Provider;

c) each FAIS_SCSI_Port may reside in multiple FAIS Regions; and

d) each FAIS_Portal may reside in no more than a single FAIS_Region.

The interpretation of the meaning of the FAIS_RegionId is outside the scope of FAIS.

5.4 Objects

5.4.1 BI

5.4.1.1 Description

A BI defines a back-end FAIS_SCSI_Port that may be associated with other FAIS objects or used to initiate SCSI requests to any remote SCSI target ports. A BI is activated through an association with a FAIS_Portal.

5.4.1.2 Characteristics

A BI:

a) is created and destroyed either explicitly by the FAIS_Client or implicitly by the FAIS_Provider;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) refers to zero or more FAIS_Portals; and

d) is referenced by zero or more BITs.

5.4.1.3 Attributes

See 9.2.1.

5.4.2 BIT

5.4.2.1 Description

A BIT defines a back-end I_T nexus between a BI and a remote SCSI target port that may be associated with other FAIS objects or used to initiate SCSI requests to the specified remote SCSI target port.

5.4.2.2 Characteristics

A BIT:

a) is created and destroyed either explicitly by the FAIS_Client or implicitly by the FAIS_Provider;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) refers to a single BI; and

d) is referenced by zero or more BITLs.

5.4.2.3 Attributes

See 9.2.2.

5.4.3 BITL

5.4.3.1 Description

A BITL defines a back-end I_T_L nexus between a BI and a specific LUN on a specific remote SCSI target port that may be associated with other FAIS objects or used to initiate SCSI requests to the specified remote SCSI target LUN.

5.4.3.2 Characteristics

A BITL:

a) is explicitly created and destroyed by the FAIS_Client or implicitly created and destroyed by the FAIS_Provider;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) refers to a single BIT; and

d) is referenced by zero or more BITLSetEntries.

5.4.3.3 Attributes

See 9.2.10.

5.4.4 BITLSetEntry

5.4.4.1 Description

A BITLSetEntry is a composite element of a BITLSetVDEV that associates a path in a BITLSetVDEV to a BITL.

5.4.4.2 Characteristics

A BITLSetEntry:

a) is explicitly added to and removed from a BITLSetVDEV by the FAIS_Client;

b) is identified by its index number within a BITLSetVDEV;

c) refers to a single BITL; and

d) is contained within a BITLSetVDEV.

5.4.4.3 Attributes

See 10.2.20.

5.4.5 BITLSetVDEV

5.4.5.1 Description

A BITLSetVDEV is a VDEV that contains BITLSetEntries in a multi-path relationship.

5.4.5.2 Characteristics

A BITLSetVDEV:

a) is explicitly created and destroyed by the FAIS_Client;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) contains zero or more BITLSetEntries; and

d) is referenced by zero or more FITLs or VDEVs.

5.4.5.3 Attributes

See 10.2.14.

5.4.6 BlockRange

5.4.6.1 Description

A BlockRange is a composite element of a ConcatenatedVDEV that associates a logically contiguous range of blocks in a ConcatenatedVDEV to a logically contiguous range of blocks in another VDEV.

5.4.6.2 Characteristics

A BlockRange:

a) is explicitly added to a ConcatenatedVDEV by the FAIS_Client;

b) is removed from a ConcatenatedVDEV either explicitly by the FAIS_Client or implicitly by the FAIS_Provider;

c) is identified by its LBA range within a ConcatenatedVDEV;

d) refers to a single VDEV; and

e) is contained within a ConcatenatedVDEV.

5.4.6.3 Attributes

See 10.2.18.

5.4.7 Column

5.4.7.1 Description

A Column is a composite element of a StripedVDEV that associates a stripe column in a StripedVDEV to another VDEV.

5.4.7.2 Characteristics

A Column:

a) is explicitly added to and removed from a StripedVDEV by the FAIS_Client;

b) is identified by its index number within a StripedVDEV;

c) refers to a single VDEV; and

d) is contained within a StripedVDEV.

5.4.7.3 Attributes

See 10.2.16.

5.4.8 ConcatenatedVDEV

5.4.8.1 Description

A ConcatenatedVDEV is a VDEV that contains BlockRanges in a concatenation relationship.

5.4.8.2 Characteristics

A ConcatenatedVDEV:

a) is explicitly created and destroyed by the FAIS_Client;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) contains zero or more non-overlapping BlockRanges; and

d) is referenced by zero or more FITLs or VDEVs.

5.4.8.3 Attributes

See 10.2.5.

5.4.9 FAIS_Portal

5.4.9.1 Description

A FAIS_Portal provides access to the SCSI transport for a FAIS_SCSI_Port. An FT or BI is activated when they are associated with a FAIS_Portal.

5.4.9.2 Characteristics

A FAIS_Portal:

a) is created and destroyed either explicitly by the FAIS_Client or implicitly by the FAIS_Provider;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) refers to a single FAIS_Region; and

d) is referenced by zero or more FTs or BIs.

5.4.9.3 Attributes

See 7.3.7.

5.4.10 FIT

5.4.10.1 Description

A FIT defines a front-end I_T nexus between an FT and remote SCSI initiator port that may be associated with other FAIS objects.

5.4.10.2 Characteristics

A FIT:

a) is created and destroyed either explicitly by the FAIS_Client or implicitly by the FAIS_Provider;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) refers to a single FT; and

d) is referenced by zero or more FITLs.

5.4.10.3 Attributes

See 8.2.2.

5.4.11 FITL

5.4.11.1 Description

A FITL object defines a front-end I_T_L nexus between a specific LUN on an FT and a specific remote SCSI initiator port.

5.4.11.2 Characteristics

A FITL:

a) is explicitly created and destroyed by the FAIS_Client;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) refers to a single FIT; and

d) refers to zero or one VDEV.

5.4.11.3 Attributes

See 8.2.3.

5.4.12 FLUVDEV

5.4.12.1 Description

A FLUVDEV is a VDEV that defines the association between a FITL and another VDEV. The FLUVDEV does not define any I/O processing semantics that may be offloaded to the FAIS_Provider.

5.4.12.2 Characteristics

A FLUVDEV:

a) is explicitly created and destroyed by the FAIS_Client;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) refers to zero or one VDEV; and

d) is referenced by zero or more FITLs.

5.4.12.3 Attributes

See 10.2.4.

5.4.13 FT

5.4.13.1 Description

An FT defines a front-end FAIS_SCSI_Port that may be associated with other FAIS objects. An FT is activated through an association with a FAIS_Portal.

5.4.13.2 Characteristics

A FT:

a) is created and destroyed either explicitly by the FAIS_Client or implicitly by the FAIS_Provider;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) refers to zero or more FAIS_Portals; and

d) is referenced by zero or more BITs.

5.4.13.3 Attributes

See 8.2.1.

5.4.14 Mirror

5.4.14.1 Description

A Mirror is a composite element of the MirroredVDEV that associates a mirror component in a Mirrored-VDEV to another VDEV.

5.4.14.2 Characteristics

A Mirror:

a) is explicitly added to or removed from a MirroredVDEV by a FAIS_Client;

b) is identified by its index number within a MirroredVDEV;

c) refers to a single VDEV; and

d) is contained within a MirroredVDEV.

5.4.14.3 Attributes

See 10.2.17.

5.4.15 MirroredVDEV

5.4.15.1 Description

A MirroredVDEV is a VDEV that contains Mirrors in a mirroring relationship.

5.4.15.2 Characteristics

A MirroredVDEV:

a) is explicitly created and destroyed by the FAIS_Client;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) contains zero or more Mirrors; and

d) is referenced by zero or more FITLs or VDEVs.

5.4.15.3 Attributes

See 10.2.12.

5.4.16 StripedVDEV

5.4.16.1 Description

A StripedVDEV is a VDEV that contains Columns in a striping relationship.

5.4.16.2 Characteristics

A StripedVDEV:

a) is explicitly created and destroyed by the FAIS_Client;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) contains zero or more Columns; and

d) is referenced by zero or more FITLs or VDEVs.

5.4.16.3 Attributes

See 10.2.11.

5.4.17 VDEV

5.4.17.1 Description

A VDEV is an abstract class that defines the common attributes for all VDEVs.

5.4.17.2 Characteristics

A VDEV:

a) is an abstract class that is used to derive concrete VDEV types.

All VDEVs:

a) are identified by an ObjectCHandle and ObjectPHandle; and

b) are referenced by zero or more FITLs or VDEVs.

5.4.17.3 Attributes

See 10.2.3.

5.4.18 XMapEntry

5.4.18.1 Description

An XMapEntry is a composite element of an XMapVDEV that associates a logically contiguous range of blocks in an XMapVDEV, based on read and write permissions, to a logically contiguous range of blocks in another VDEV.

5.4.18.2 Characteristics

An XMapEntry:

a) is explicitly added to an XMapVDEV by the FAIS_Client;

b) is removed from an XMapVDEV either explicitly by the FAIS_Client or implicitly by the FAIS_Provider;

c) is identified by its LBA range within an XMapVDEV;

d) either refers to a single VDEV for reads and a single VDEV for writes, or it refers to a single VDEV for both reads and writes; and

e) is contained within an XMapVDEV.

5.4.18.3 Attributes

See 10.2.19.

5.4.19 XMapVDEV

5.4.19.1 Description

An XMapVDEV is a VDEV that contains XMapEntries in a concatenation relationship.

5.4.19.2 Characteristics

An XMapVDEV:

a) is explicitly created and destroyed by the FAIS_Client;

b) is identified by its ObjectCHandle and ObjectPHandle;

c) contains zero or more non-overlapping XMapEntries; and

d) is referenced by zero or more FITLs or VDEVs.

5.4.19.3 Attributes

See 10.2.13.

6 General services

6.1 Overview

General Services include those types, data structures, or functions that may be used by more than one other defined service.

6.2 Constants

FAIS_API General Services constants are defined in table 3.

Table 3 – General services constants

Constant	Description	Default Value	Implementation Value[a]
FAIS_BLOCK_SIZE	Defines the size, in bytes, of a block.	512	y
FAIS_CLIENT_NAME_LEN	Defines the maximum number of characters in the fais_Init() nickname.	255	y
FAIS_PROVIDER_CAPABILITIES_LEN	Defines the length, in bytes, of the ProviderCapabilities array.	256	N/A
FAIS_SCSI_CDBLEN_MAX	Defines the length, in bytes, of a SCSI Command Descriptor Block (CDB).	256	y
FAIS_STRLEN_MAX	Defines the maximum string length in bytes. The length does not include the string termination character FAIS_STR_TERM_CHAR.	255	y
FAIS_STR_TERM_CHAR	Defines the value used to terminate strings used in FAIS.	0	y
FAIS_VENDOR_ID_LEN	Defines the length, in bytes, of the T10 Vendor ID.	8	n
FAIS_PROVIDER_VERSION_LEN	Defines the length, in bytes, of version numbers used in FAIS.	FAIS_STRLEN_MAX	n
IN[b]	This definition allows IN to be used in FAIS to indicate input values.	N/A[c]	N/A
OUT[b]	This definition allows OUT to be used in FAIS to indicate output values.	N/A[d]	N/A
INOUT[b]	This definition allows INOUT to be used in FAIS to indicate input/output values.	N/A[e]	N/A

[a] The implementation value column indicates whether an implementation is allowed to change the value from the default.

[b] For pointers, the IN, OUT or INOUT attribute applies to the data referenced by the pointer.

[c] In C this is defined as '#define IN' (i.e., an empty define).

[d] In C this is defined as '#define OUT' (i.e., an empty define).

[e] In C this is defined as '#define INOUT' (i.e., an empty define).

6.3 Data structures

6.3.1 FAIS_ObjectType

6.3.1.1 Format

```
typedef enum FAIS_ObjectType_E {
        FAIS_OBJECTTYPE_BI = 0,
        FAIS_OBJECTTYPE_BIT = 1,
        FAIS_OBJECTTYPE_BITL = 2,
        FAIS_OBJECTTYPE_FT = 3,
        FAIS_OBJECTTYPE_FIT = 4,
        FAIS_OBJECTTYPE_FITL = 5,
        FAIS_OBJECTTYPE_PORTAL = 6,
        FAIS_OBJECTTYPE_VDEV = 7
} FAIS_ObjectType_T;
```

6.3.1.2 Description

The types of objects supported by FAIS.

6.3.1.3 Values

FAIS_OBJECTTYPE_BI: See 5.4.1.

FAIS_OBJECTTYPE_BIT: See 5.4.2.

FAIS_OBJECTTYPE_BITL: See 5.4.3.

FAIS_OBJECTTYPE_FT: See 5.4.13.

FAIS_OBJECTTYPE_FIT: See 5.4.10.

FAIS_OBJECTTYPE_FITL: See 5.4.11.

FAIS_OBJECTTYPE_PORTAL: See 5.4.9.

FAIS_OBJECTTYPE_VDEV: See 5.4.17.

6.3.2 FAIS_ClientRequest_Header

6.3.2.1 Format

```
typedef struct FAIS_ClientRequest_Header_S {
    IN  void (*pCbFunction) (
            FAIS_Status_T status,
            void *pRequest);
    IN  void *cbContext;
    IN  int32 timeout;
    OUT FAIS_ClientCHandle_T ClientCHandle;
    IN  FAIS_ClientPHandle_T ClientPHandle;
    IN  uint32 VendorSpecificLength;
    INOUTuint8 *pVendorSpecific;
} FAIS_ClientRequest_Header_T;
```

6.3.2.2 Description

Common header for the function parameter blocks passed in asynchronous-capable request methods called by the FAIS_Client. See also 4.5.3.

6.3.2.3 Members

pCbFunction: A pointer to the callback function that the FAIS_Provider shall call when a successfully submitted asynchronous request completes. The completion status and the function parameter block are passed as arguments to this function upon completion of the operation. The same parameter block that was specified in the asynchronous request (i.e., same memory address) is returned in the callback function. If pCbFunction is NULL, the request is performed synchronously.

cbContext: Opaque data that is initialized by the FAIS_Client and is not modified by the FAIS_Provider. This field may be used as context data for the FAIS_Client to associate a completion callback with the corresponding request.

timeout: The maximum amount of time allowed for the operation to complete. The timeout value shall be as shown in table 4.

Table 4 – Valid Timeout values

Value	Description	Completion
>=1	Allow the request to wait up to the specified number of milliseconds.	If the timeout value is exceeded, the request shall fail with FAIS_STATUS_TIMED_OUT.
0	Do not allow the request to wait.	If the operation is unable to complete immediately, the request shall fail with FAIS_STATUS_WOULD_BLOCK.
-1	Allow the request to wait indefinitely.	The operation shall not fail due to the timeout value.
< -1	Reserved	

ClientCHandle: The handle assigned by the FAIS_Client during fais_Init() for this caller.

ClientPHandle: The handle assigned by the FAIS_Provider during fais_Init() for this caller.

VendorSpecificLenth: The length, in bytes, of the Vendor Specific information. This field shall be set to zero if no Vendor Specific information is available.

pVendorSpecific: A pointer to Vendor Specific information. This field shall be set to NULL if no Vendor Specific information is available.

NOTE A FAIS_Platform is not required to support vendor specific fields. If vendor specific fields are not recognized, the function call should be completed as if no vendor specific fields were specified.

6.3.3 FAIS_ObjectCHandle

6.3.3.1 Format

```
typedef void * FAIS_ObjectCHandle_T;
```

6.3.3.2 Description

The FAIS_ObjectCHandle is an opaque handle assigned by the FAIS_Client to identify a FAIS object. The FAIS_ObjectCHandle is passed by the FAIS_Provider when the FAIS object is referenced in a call to a FAIS_Client (e.g., completion callback).

6.3.4 FAIS_ObjectPHandle

6.3.4.1 Format

```
typedef void * FAIS_ObjectPHandle_T;
```

6.3.4.2 Description

The FAIS_ObjectPHandle is an opaque handle assigned by the FAIS_Provider to identify a FAIS object. The FAIS_ObjectPHandle is passed by the FAIS_Client when the object is referenced in a call to a FAIS_Provider (e.g., function call).

6.3.5 FAIS_ClientCHandle

6.3.5.1 Format

```
typedef void * FAIS_ClientCHandle_T;
```

6.3.5.2 Description

The FAIS_ClientCHandle is an opaque handle assigned by the FAIS_Client to identify itself. The FAIS_ClientCHandle is passed by the FAIS_Provider when it calls a FAIS_Client (e.g., completion callback).

6.3.6 FAIS_ClientPHandle

6.3.6.1 Format

```
typedef void * FAIS_ClientPHandle_T;
```

6.3.6.2 Description

The FAIS_ClientPHandle is an opaque handle assigned by the FAIS_Provider to identify the FAIS_Client. The FAIS_ClientPHandle is passed by the FAIS_Client when it calls a FAIS_Provider (e.g., function call).

6.3.7 FAIS_HandleSet

6.3.7.1 Format

```
typedef struct FAIS_HandleSet_S {
        FAIS_ObjectCHandle_T ObjCHandle;
        FAIS_ObjectPHandle_T ObjPHandle;
        FAIS_ClientCHandle_T ClientCHandle;
        FAIS_ClientPHandle_T ClientPHandle;
} FAIS_HandleSet_T;
```

6.3.7.2 Description

Data structure containing all handles associated with an object.

6.3.7.3 Members

ObjCHandle: See 6.3.3.

ObjPHandle: See 6.3.4.

ClientCHandle: See 6.3.5.

ClientPHandle: See 6.3.6.

6.3.8 FAIS_LUN

6.3.8.1 Format

```
typedef uint64 FAIS_LUN_T;
```

6.3.8.2 Description

Store SCSI LUN in the format as defined in SAM-3. Byte zero of a SCSI LUN shall be stored at the lowest address byte in FAIS_LUN and successive bytes of the SCSI LUN shall be stored at successively higher address bytes of FAIS_LUN.

6.3.9 FAIS_Status

6.3.9.1 Format

```
typedef enum FAIS_Status_E {
        FAIS_STATUS_SUCCESS = 0,
        FAIS_STATUS_OBJECT_NOT_FOUND = 1,
        FAIS_STATUS_INSUFFICIENT_RESOURCES = 2,
        FAIS_STATUS_INVALID_ARGUMENT = 3,
        FAIS_STATUS_OBJECT_EXISTS = 4,
        FAIS_STATUS_NOT_ACCESSIBLE = 5,
        FAIS_STATUS_ACCESS_DENIED = 6,
        FAIS_STATUS_RANGE_ERROR = 7,
        FAIS_STATUS_BUSY = 8,
        FAIS_STATUS_ALREADY_EXISTS = 9,
        FAIS_STATUS_TIMED_OUT = 10,
        FAIS_STATUS_NOT_SUPPORTED = 11,
        FAIS_STATUS_NOT_FOUND = 12,
        FAIS_STATUS_NO_SPACE = 13,
        FAIS_STATUS_CONNECTION_REFUSED = 14,
        FAIS_STATUS_WOULD_BLOCK = 15,
        FAIS_STATUS_CANCELED = 16,
        FAIS_STATUS_INVALID_CLIENT = 17,
        FAIS_STATUS_NO_STATS = 18,
} FAIS_Status_T;
```

6.3.9.2 Description

The enumeration FAIS_Status_E defines status conditions available from FAIS.

6.3.9.3 Values

FAIS_STATUS_SUCCESS: Normal acceptance or completion.

FAIS_STATUS_OBJECT_NOT_FOUND: FAIS Object does not exist.

FAIS_STATUS_INSUFFICIENT_RESOURCES: Unable to complete or accept due to resource limitation.

FAIS_STATUS_INVALID_ARGUMENT: A function argument is invalid.

FAIS_STATUS_OBJECT_EXISTS: FAIS Object does exist.

FAIS_STATUS_ACCESS_DENIED: Access denied.

FAIS_STATUS_RANGE_ERROR: The request exceeds the boundaries of the specified object or resource.

FAIS_STATUS_BUSY: There is already an outstanding operation on the requested object or resource.

FAIS_STATUS_ALREADY_EXISTS: The specified object or resource already exists.

FAIS_STATUS_TIMED_OUT: The operation timed out.

FAIS_STATUS_NOT_SUPPORTED: The operation is not supported.

FAIS_STATUS_NOT_FOUND: The specified object or resource was not found.

FAIS_STATUS_ACCESS_DENIED: The caller does not have sufficient privileges.

FAIS_STATUS_NOT_ACCESSIBLE: The specified object or information is not accessible.

FAIS_STATUS_NO_SPACE: There is insufficient capacity or resources to complete the request (e.g., out of memory, container object is full).

FAIS_STATUS_CONNECTION_REFUSED: The remote endpoint refused to establish the connection.

FAIS_STATUS_WOULD_BLOCK: The requested non-blocking operation would block.

FAIS_STATUS_CANCELLED: The requested operation was cancelled.

FAIS_STATUS_INVALID_CLIENT: The specified FAIS_ClientPHandle is invalid.

FAIS_STATUS_NO_STATS: No statistics supported for this object.

6.3.10 FAIS_ProviderInfo

6.3.10.1 Format

```
typedef struct FAIS_ProviderInfo_S {
        uint8 VendorId[FAIS_VENDOR_ID_LEN];
        uint8 ProviderVersion[FAIS_PROVIDER_VERSION_LEN];
        uint8 ProviderCapabilities[FAIS_PROVIDER_CAPABILITIES_LEN];
        uint32 APIVersion;
} FAIS_ProviderInfo_T;
```

6.3.10.2 Description

Data structure for determining the version level of the API and the vendor version information.

6.3.10.3 Members

VendorID: A fixed length array of ASCII characters that uniquely identifies the vendor (see 3.10).

ProviderVersion: A fixed length array of ASCII characters that identifies the FAIS_Provider version number. Each FAIS_Provider supplies vendor specific values. Unused characters shall be filled with ASCII space characters (20h).

ProviderCapabilities: A fixed length array of bytes that identifies vendor specific FAIS_Provider capabilities information.

APIVersion: A fixed length array of ASCII characters that identifies the version number of the FAIS standard with which the FAIS_Provider was compiled. The value shall be as shown in table 5.

Table 5 – APIVersion Values

Value	Version
1	This standard.
all others	Reserved

6.3.11 FAIS_EnumerationFilter

6.3.11.1 Format

```
typedef uint32 FAIS_EnumerationFilter_T;
```

6.3.11.2 Description

A filter for enumerating objects.

Table 6 – FAISEnumerationFilter_T values

Enumeration Filter	Description	Value (hex)
FAIS_ENUMERATE_ALL	Enumerate all objects.	0
Reserved		00000001 .. 7FFFFFFF
VENDOR_SPECIFIC	Vendor Specific enumerator values.	80000000 .. FFFFFFFF

6.3.12 FAIS_IO_Stats_T

6.3.12.1 Format

```
typedef struct FAIS_IO_Stats_S {
        uint64  ReadIOs;
        uint64  BlocksRead;
        uint64  WriteIOs;
        uint64  BlocksWritten;
} FAIS_IO_Stats_T;
```

6.3.12.2 6.3.12.2 Description

Data structure for returning statistics on a VDEV, FITL, or BITL.

6.3.12.3 Members

ReadIOs: The cumulative count of all reads. A value of zero may mean the statistic is not available.

BlocksRead: The cumulative count of data read in units of blocks. A value of zero may mean the statistic is not available.

WriteIOs: The cumulative count of all writes. A value of zero may mean the statistic is not available.

BlocksWritten: The cumulative count of data written in units of blocks. A value of zero may mean the statistic is not available.

6.4 Function Calls

6.4.1 fais_Init

6.4.1.1 Format

```
FAIS_Status_T fais_Init (
    IN  uint8 * ClientNickName,
    IN  uint8 Flags,
    OUT FAIS_ProviderInfo_T * VersionPtr,
    IN  FAIS_ClientCHandle_T ClientCHandle,
    OUT FAIS_ClientPHandle_T * pClientPHandle,
    IN  uint32 VendorSpecificLength,
    INOUTuint8 *pVendorSpecific)
```

6.4.1.2 Description

Initialize a FAIS_Client's access to a FAIS_Provider. This function shall be completed successfully before any other FAIS_API function may be called.

6.4.1.3 Arguments

ClientNickName: A character string containing a name of the FAIS_Client created by the FAIS_Client. The length is FAIS_CLIENT_NAME_LEN. Unused characters shall contain ASCII space characters (20h). Usage of this field is outside the scope of this standard.

Flags: A bit field of flags that specify fais_Init alternative behaviors. FAIS_Providers that do not hold events shall support the FAIS_EVENTS_HOLD_ALL value 0. FAIS_Providers that hold events shall support the FAIS_EVENTS_HOLD_ALL value 1. At least one of the FAIS_EVENTS_HOLD_ALL values shall be supported. Valid bit values for the Flags field are shown in table 7.

Table 7 – fais_Init Flag bits

Bit(s)	Flag	Description
0	FAIS_EVENTS_HOLD_ALL	If set to 1, the FAIS_Provider shall hold all events when a condition exists for which events cannot be delivered. If set to 0, the FAIS_Provider shall not hold events.
1 .. 7	Reserved	

VersionPtr: A pointer to the FAIS_Provider version information, produced by the FAIS_Provider and returned to the FAIS_Client.

ClientCHandle: A FAIS_Client generated handle, to be used in other FAIS_Provider callbacks and events for this instance. The handle may be different each time fais_Init() is used.

pClientPHandle: A pointer to the FAIS_Provider generated handle, to be used in other FAIS_Client calls for this instance. The handle may be different each time fais_Init() is used.

VendorSpecificLenth: The length, in bytes, of the Vendor Specific information. This field shall be set to zero if no Vendor Specific information is available.

pVendorSpecific: A pointer to Vendor Specific information. This field shall be set to NULL if no Vendor Specific information is available.

NOTE A FAIS_Platform is not required to support vendor specific fields. If vendor specific fields are not recognized, the function call should be completed as if no vendor specific fields were specified.

6.4.1.4 Return values

Returns FAIS_STATUS_SUCCESS upon successful completion of the synchronous function call. Otherwise, returns the status condition identifying the error.

6.4.2 fais_DeInit

6.4.2.1 Format

```
FAIS_Status_T fais_DeInit (
    IN  FAIS_ClientPHandle_T ClientPHandle,
    IN  uint32 Flags,
    IN  uint32 VendorSpecificLength,
    INOUTuint8 *pVendorSpecific)
```

6.4.2.2 Description

This function deinitializes the FAIS_Client's access to the FAIS_Provider. After a successful call to this function, the only valid function that the FAIS_Client may subsequently call is fais_Init(). If the FAIS_OBJECTS_PERSIST flag is set, any objects that have been registered by that FAIS_Client with the DPC still exists in the DPC, otherwise all objects registered by that FAIS_Client are removed.

6.4.2.3 Arguments

ClientPHandle: A unique identifier of the FAIS_Client that is no longer to be registered.

Flags: A bit field of flags that specify fais_DeInit alternative behaviors. FAIS_Providers that do not retain objects shall support the FAIS_OBJECTS_PERSIST value 0. FAIS_Providers that retain objects after the fais_DeInit function is called shall support the FAIS_OBJECTS_PERSIST value 1. At least one of the FAIS_OBJECTS_PERSIST values shall be supported. At least one of the FAIS_CALLBACKS_CANCEL values shall be supported that define callback cancellation behavior. Valid bit values for the Flags field are shown in table 8.

Table 8 – fais_DeInit flag bits

Bit	Flag	Desctription
0	FAIS_OBJECTS_PERSIST	If set to 1, the FAIS_Provider shall retain all FAIS Objects that have been registered by the calling FAIS_Client when fais_DeInit is invoked. If set to 0, the FAIS_Provider shall not retain any FAIS Object that have been registered by the calling FAIS_Client when fais_DeInit is invoked.
1	FAIS_CALLBACKS_CANCEL	If set to 1, fais_DeInit shall cancel all callbacks and shall not succeed until all callbacks are cancelled. If set to 0, fais_DeInit shall not cancel any callback and shall not succeed until all callbacks are completed.
2 .. 31	Reserved	

VendorSpecificLenth: The length, in bytes, of the Vendor Specific information. This field shall be set to zero if no Vendor Specific information is available.

pVendorSpecific: A pointer to Vendor Specific information. This field shall be set to NULL if no Vendor Specific information is available.

NOTE A FAIS_Platform is not required to support vendor specific fields. If vendor specific fields are not recognized, the function call should be completed as if no vendor specific fields were specified.

6.4.2.4 Return values

Returns FAIS_STATUS_SUCCESS upon successful completion of the synchronous function call. Otherwise, returns the status condition identifying the error.

6.4.3 fais_Object_Enumerate

6.4.3.1 Format

```
typedef struct FAIS_Object_Enumerate_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  FAIS_ObjectType_T type;
    IN  uint32 nStart;
    IN  FAIS_EnumerationFilter_T filter;
    IN  uint32 nObjectsMax;
    OUT uint32 nObjectsActual;
    OUT FAIS_HandleSet_T *pObjects;
} FAIS_Object_Enumerate_T;

FAIS_Status_T fais_Object_Enumerate (FAIS_Object_Enumerate_T * ptr);
```

6.4.3.2 Description

Query for FAIS objects in the system.

6.4.3.3 Arguments

ptr: A pointer to a FAIS_Object_Enumerate_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous functions. See 6.3.2.

type: The type of objects to enumerate.

nStart: The index of the first object to be placed in the pObjects buffer. Objects are numbered starting from 0.

filter: Filter for enumerating the objects.

nObjectsMax: The maximum number of objects for which the FAIS_Client is able to enumerate (i.e., as limited by the buffer referenced by pObjects). If set to 0, no objects are returned and the pObjects argument is ignored.

nObjectsActual: The actual number of objects available for enumeration.

pObjects: A pointer to a buffer for storing the handles of the objects that were found. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nObjectsMax. This argument is optional, and is ignored if the nObjectsMax argument is 0.

6.4.3.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

6.4.4 fais_ObjectHandle_Update

6.4.4.1 Format

```
FAIS_Status_T fais_ObjectHandle_Update (
```

```
    IN  FAIS_ObjectPHandle_T ObjectPHandle,
    IN  FAIS_ObjectCHandle_T ObjectCHandle)
```

6.4.4.2 Description

A method that allows a FAIS_Client to assign a FAIS_ObjectCHandle for an implicitly created object, or to change it for any object (e.g., another FAIS_Client nondisruptively takes over management of an existing object).

6.4.4.3 Arguments

ObjectPHandle: See 6.3.4.

ObjectCHandle: See 6.3.3.

6.4.4.4 Return values

Returns FAIS_STATUS_SUCCESS upon success. Otherwise, returns the status condition identifying the error.

6.4.5 fais_Get_IO_Stats

6.4.5.1 Format

```
FAIS_Status_T fais_Get_IO_Stats(
    IN  FAIS_ObjectPHandle_T ObjPHandle,
    IN  FAIS_ClientCHandle_T ClientCHandle,
    IN  FAIS_ClientPHandle_T ClientPHandle,
    IN  char   ClearOnRead,
    OUT FAIS_IO_Stats_T *IOStats);
```

6.4.5.2 Description

Retrieves the IO statistics for the requested object.

6.4.5.3 Arguments

ObjPHandle: The handle for the object containing the statistics, and assigned by the FAIS_Provider.

ClientCHandle: The handle assigned by the FAIS_Client during fais_init() for this client.

ClentPHandle: The handle assigned by the FAIS_Provider during fais_init() for this client.

ClearOnRead: If non-zero, counters shall be cleared on read.

IOStats: A pointer to a buffer for storing the IO statistics for the requested object.

6.4.5.4 Return values

Returns FAIS_STATUS_SUCCESS upon successful completion of the synchronous function. Otherwise, returns the status condition identifying the error.

7 Port services

7.1 Overview

Port Services provide support for creating, removing and performing operations on FAIS_SCSI_Ports.

7.2 Constants

FAIS_API Port Services constants are defined in table 9.

Table 9 – Port services constants

Constant	Description	Value
FAIS_PORTNAME_ISCSI_ISID_MAX	The maximum length, in bytes, of the iSCSI ISID.	15
FAIS_PORTNAME_ISCSI_TPGT_MAX	The maximum length, in bytes, of the iSCSI TPGT.	7
FAIS_NODENAME_ISCSI_MAX	The maximum length, in bytes, of the iSCSI Nodename.	224
FAIS_REGION_ID_MAX	The maximum length, in bytes, of the identifier for a FAIS_Region.	255

7.3 Data structures

7.3.1 FAIS_Protocol

7.3.1.1 Format

```
typedef enum FAIS_Protocol_E {
        FAIS_PROTOCOL_FCP = 1,
        FAIS_PROTOCOL_ISCSI = 2
} FAIS_Protocol_T;
```

7.3.1.2 Description

The transport protocols supported by FAIS.

7.3.1.3 Values

FAIS_PROTOCOL_FCP: Transport Protocol is FCP (see FCP-3).

FAIS_PROTOCOL_ISCSI: Transport Protocol is iSCSI (see RFC 3720).

7.3.2 FAIS_IPVERSION

7.3.2.1 Format

```
typedef enum FAIS_IPVersion_E {
        FAIS_IPVERSION_6 = 6,
        FAIS_IPVERSION_4 = 4
} FAIS_IPVersion_T;
```

7.3.2.2 Description

The Internet Protocol versions supported by FAIS.

7.3.2.3 Values

FAIS_IPVERSION_6: Internet Protocol version 6 (IPv6) (see RFC 2460).

FAIS_IPVERSION_4: Internet Protocol version 4 (IPv4) (see RFC 791).

7.3.3 FAIS_PortName_FCP

7.3.3.1 Format

```
typedef FAIS_PortName_FCP_S {
        uint8NodeName[8];
        uint8PortName[8];
} FAIS_PortName_FCP_T;
```

7.3.3.2 Description

Data structure for the worldwide unique name of a FAIS_SCSI_Port in FCP.

7.3.3.3 Members

NodeName: The value of the Node_Name (see FC-FS-2). The most significant byte of the Node_Name shall be stored in byte 0, and the least sigifiicant byte shall be stored in byte 7.

PortName: The value of the N_Port_Name (see FC-FS-2). The most significant byte of the Node_Name shall be stored in byte 0, and the least sigifiicant byte shall be stored in byte 7.

7.3.4 FAIS_PortName_ISCSI

7.3.4.1 Format

```
typedef struct FAIS_PortName_ISCSI_S {
        uint8 NodeName[FAIS_NODENAME_ISCSI_MAX];
        union {
            uint8 Isid[FAIS_PORTNAME_ISCSI_ISID_MAX];
            uint8 Tpgt[FAIS_PORTNAME_ISCSI_TPGT_MAX];
        } PortSuffix;
} FAIS_PortName_ISCSI_T;
```

7.3.4.2 Description

Data structure for the worldwide unique name of a FAIS_SCSI_Port in iSCSI.

7.3.4.3 Members

NodeName: The UTF-8 encoding of the hexidecimal represntation of the node name (e.g., iSCSI name) including the '0x' prefix and the terminating NULL.

PortSuffix: The UTF-8 encoding of the hexidecimal representaton of the ISID if operating as an initiator, or the TPGT if operating as a target. This includes the '0x' prefix, and the terminating NULL. The specific union member that is used is dependent upon the usage of this data structure (e.g., TPGT is used if this data structure is contained in an FT).

7.3.5 FAIS_PortName

7.3.5.1 Format

```
typedef struct FAIS_PortName_S {
        FAIS_Protocol_T Protocol;
        union {
            FAIS_PortName_FCP_T FCPName;
            FAIS_PortName_ISCSI_T ISCSIName;
        } Name;
```

```
} FAIS_PortName_T;
```

7.3.5.2 Description

Data structure for the worldwide unique name of the FAIS_SCSI_Port. The name is defined by SAM, but its size and format specified by its respective transport protocol.

7.3.5.3 Members

Protocol: The transport protocol of the FAIS_SCSI_Port.

Name: The protocol-dependent name of the FAIS_SCSI_Port.

7.3.6 FAIS_RegionId

7.3.6.1 Format

```
typedef struct FAIS_RegionId_S {
        uint8 Data[FAIS_REGION_ID_MAX];
} FAIS_RegionId_T;
```

7.3.6.2 Description

Data structure of the identifier for a FAIS_Region.

7.3.6.3 Members

Data: This is FAIS_Provider-dependent.

7.3.7 FAIS_Portal

7.3.7.1 Format

```
typedef struct FAIS_Portal_S {
        FAIS_ObjectCHandle_T ObjCHandle;
        FAIS_RegionId_T RegionId;
        FAIS_Portal_id_T Portal_id;
} FAIS_Portal_T;
```

7.3.7.2 Description

Data structure for a FAIS_Portal. See also 5.4.9.

7.3.7.3 Members

ObjCHandle: Handle assigned by the FAIS_Client to this object.

RegionId: Identifier of the FAIS_Region associated with this FAIS_Portal.

Portal_id: Identifier of this FAIS_Portal.

7.3.8 FAIS_Portal_id_FCP

7.3.8.1 Format

```
typedef uint32 FAIS_Portal_id_FCP_T;
```

7.3.8.2 Description

Data structure of the identifier for a Transport_Port in FCP. The Fibre Channel Port Address is stored in the lowest order 3 bytes of `FAIS_Portal_id_FCP_T variable` and in network byte order.

7.3.9 FAIS_Portal_id_ISCSI

7.3.9.1 Format

```
typedef struct FAIS_Portal_id_ISCSI_S {
        FAIS_IPVersion_T IPVersion;
        union {
            uint8 IPv6Address[16];
            uint32 IPv4Address;
        } Address;
        uint16 TCPPort;
} FAIS_Portal_id_ISCSI_T;
```

7.3.9.2 Description

Data structure of the identifier for a Transport_Port in iSCSI. If the transport protocol is iSCSI, Discovery-specific commands shall be supported by the FAIS_Portal_id_ISCSI_T.

7.3.9.3 Members

IPVersion: Internet Protocol version that defines the structure of the IP address.

Address: This field contains the IP address for either IPv4 or IPv6. The IP address is stored in network byte order.

TCPPort: TCP Port of the Transport_Port. The TCPPort field is stored in network byte order.

7.3.10 FAIS_Portal_id

7.3.10.1 Format

```
typedef struct FAIS_Portal_id_S {
        FAIS_Protocol_T Protocol;
        union {
            FAIS_Portal_id_FCP_T FCPId;
            FAIS_Portal_id_ISCSI_T ISCSIId;
        } Id;
} FAIS_Portal_id_T;
```

7.3.10.2 Description

Data structure of the identifier for a Transport_Port. A Transport_Port is an addressable endpoint in the network. The size and format of the identifier is defined by the transport protocol.

7.3.10.3 Members

Protocol: The transport protocol of the Transport_Port.

Id: The protocol-dependent identifier of the Transport_Port.

7.4 Function Calls

7.4.1 fais_Region_Enumerate

7.4.1.1 Format

```
typedef struct FAIS_Region_Enumerate_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  uint32 nStart;
    IN  FAIS_EnumerationFilter_T filter;
    IN  uint32 nRegionsMax;
    OUT uint32 nRegionsActual;
    OUT FAIS_RegionId_T * pRegions;
} FAIS_Region_Enumerate_T;

FAIS_Status_T fais_Region_Enumerate (FAIS_Region_Enumerate_T * ptr);
```

7.4.1.2 Description

Query for the FAIS_Regions in the system.

7.4.1.3 Arguments

ptr: A pointer to a FAIS_Region_Enumerate_T structure, the fields of which are defined in this sub-clause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

nStart: The index of the first object to be placed in the pRegions buffer. Regions are numbered starting from 0.

filter: Filter for enumerating regions.

nRegionsMax: The maximum number of FAIS_Regions for which the FAIS_Client is able to enumerate (i.e., as limited by the buffer referenced by pRegions). If set to 0, no FAIS_Regions are returned and the pRegions argument is ignored.

nRegionsActual: The actual number of FAIS_Regions available for enumeration.

pRegions: A pointer to a buffer for storing the identifiers for the FAIS_Regions that were found. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nRegions.

7.4.1.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

7.4.2 fais_Region_GetStatus

7.4.2.1 Format

```
typedef struct FAIS_Region_GetStatus_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  FAIS_RegionId_T RegionId;
    IN  FAIS_EnumerationFilter_T filter;
    INOUTuint32 nPortals;
    OUT FAIS_HandleSet_T * pPortals;
} FAIS_Region_GetStatus_T;
```

```
FAIS_Status_T fais_Region_GetStatus (FAIS_Region_GetStatus_T * ptr);
```

7.4.2.2 Description

Get status on a specific FAIS_Region.

7.4.2.3 Arguments

ptr: A pointer to a FAIS_Region_GetStatus_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

RegionId: The identifier of FAIS_Region to query.

filter: Filter for enumerating Portals.

nPortals: The number of FAIS Portals. On input, the FAIS_Client specifies the maximum number of FAIS Portals it is able to query for (i.e., size of buffer referenced by the pPortals argument). If set to 0 on input, no FAIS Portals are returned and the pPortals argument is ignored. On output, the FAIS_Provider updates this to indicate the actual number of FAIS Portals found.

pPortals: A pointer to a buffer for storing the FAIS handles for the FAIS Portals that were found. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nPortals. This argument is optional and is ignored if the nPortals argument is 0.

7.4.2.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

7.4.3 fais_Region_GetDetail

7.4.3.1 Format

```
typedef struct FAIS_Region_GetDetail_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  FAIS_RegionId_T RegionId;
    INOUTuint32 nRegionDetailBytes;
    OUT void *pRegionDetailBytes;
} FAIS_Region_GetDetail_T;

FAIS_Status_T fais_Region_GetDetail (FAIS_Region_GetDetail_T * ptr);
```

7.4.3.2 Description

Get FAIS_Platform-specific attributes for the specified FAIS_Region.

7.4.3.3 Arguments

ptr: A pointer to a FAIS_Region_GetDetail_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

RegionId: The identifier of FAIS_Region to query.

nRegionDetailBytes: The number of bytes describing the FAIS_Platform specific FAIS_Region details. On input, the FAIS_Client specifies the maximum number of bytes it is able to receive (i.e., as lim-

ited by the buffer referenced by pRegionDetailBytes). If set to 0 on input, no FAIS_Region details are returned and the pRegionDetailBytes argument is ignored. On output, the FAIS_Provider updates this to indicate the number of bytes actually required to return the FAIS_Region details.

pRegionDetailBytes: A pointer to a buffer for receiving the FAIS_Region details. The buffer shall be large enough to support up to the number of bytes specified by the FAIS_Client in nRegionDetailBytes. This argument is optional, and is ignored if the nRegionDetailBytes argument is 0.

7.4.3.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

7.4.4 fais_Region_SetDetail

7.4.4.1 Format

```
typedef struct FAIS_Region_SetDetail_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  FAIS_RegionId_T RegionId;
    IN  uint32 nRegionDetailBytes;
    IN  void *pRegionDetailBytes;
} FAIS_Region_SetDetail_T;

FAIS_Status_T fais_Region_SetDetail (FAIS_Region_SetDetail_T * ptr);
```

7.4.4.2 Description

Set FAIS_Platform-specific attributes for the specified FAIS_Region.

7.4.4.3 Arguments

ptr: A pointer to a FAIS_Region_SetDetail_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

RegionId: The identifier of FAIS_Region to modify.

nRegionDetailBytes: The number of bytes describing the FAIS_Platform-specific FAIS_Region details. The FAIS_Client specifies the number of bytes in the buffer referenced by pRegionDetailBytes.

pRegionDetailBytes: A pointer to a buffer for sending the FAIS_Region details. The buffer needs to be large enough to support up to the number of bytes specified by the FAIS_Client in nRegionDetailBytes. Note that the supplied FAIS_Region details completely replace the details previously specified for the FAIS_Region.

7.4.4.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

7.4.5 fais_Portal_Create

7.4.5.1 Format

```
typedef struct FAIS_Portal_Create_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectPHandle_T ObjPHandle;
```

```
    IN  uint32 flags;
    INOUTFAIS_Portal_T Portal;
} FAIS_Portal_Create_T;

FAIS_Status_T fais_Portal_Create (FAIS_Portal_Create_T * ptr);
```

7.4.5.2 Description

Creates a FAIS_Portal.

7.4.5.3 Arguments

ptr: A pointer to a FAIS_Portal_Create_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

flags: A bit field with zero or more of the flags shown in table 10.

Table 10 – fais_Portal_Create flags

Bit(s)	Flag	Description
0	FAIS_PORTAL_ID_AUTO	The identifier of the new FAIS_Portal shall be assigned automatically.
1	FAIS_REGION _AUTO	The FAIS_Region associated with the new FAIS_Portal shall be selected automatically.
2 - 31	Reserved	

Portal: The attributes of the new FAIS_Portal. On input, these shall be initialized by the FAIS_Client, as described by 7.3.7, with the following exceptions:

a) **RegionId:** On input, the FAIS_Client shall initialize this argument to the identifier of a specific FAIS_Region with which to associate the FAIS_Portal being created, if FAIS_REGION_AUTO is not set; otherwise, this is ignored on input. On output, the FAIS_Provider shall initialize this argument to the identifier of the FAIS_Region that was assigned to the created FAIS_Portal. If the FAIS_Provider was unable to assign a FAIS_Region, the operation shall fail with the appropriate FAIS_STATUS_T (see 6.3.9); and

b) **Portal_id:** On input, the FAIS_Client shall initialize this argument to an identifier for the FAIS_Portal being created, if FAIS_PORTAL_ID_AUTO is not set; otherwise, this is ignored on input. On output, the FAIS_Provider shall initialize this argument to the identifier assigned to the created FAIS_Portal. If the FAIS_Provider was unable to assign an identifier, the operation shall fail with the appropriate FAIS_STATUS_T (see 6.3.9).

7.4.5.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

7.4.6 fais_Portal_Destroy

7.4.6.1 Format

```
typedef struct FAIS_Portal_Destroy_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT  FAIS_ObjectCHandle_T ObjCHandle;
    IN   FAIS_ObjectPHandle_T ObjPHandle;
```

```
} FAIS_Portal_Destroy_T;

FAIS_Status_T fais_Portal_Destroy (FAIS_Portal_Destroy_T * ptr);
```

7.4.6.2 Description

Destroys a FAIS_Portal.

7.4.6.3 Arguments

ptr: A pointer to a FAIS_Portal_Destroy_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

7.4.6.4 Return values

FAIS_STATUS_SUCCESS upon success. Otherwise, returns the status condition identifying the error.

7.4.7 fais_Portal_GetStatus

7.4.7.1 Format

```
typedef struct FAIS_Portal_GetStatus_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    OUT FAIS_Portal_T Portal;
    IN  FAIS_EnumerationFilter_T filter;
    INOUTuint32 nFTs;
    OUT FAIS_HandleSet_T *pFTs;
    INOUTuint32 nBIs;
    OUT FAIS_HandleSet_T *pBIs;
} FAIS_Portal_GetStatus_T;

FAIS_Status_T fais_Portal_GetStatus (FAIS_Portal_GetStatus_T * ptr);
```

7.4.7.2 Description

Gets the status of a FAIS_Portal.

7.4.7.3 Arguments

ptr: A pointer to a FAIS_Portal_GetStatus_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

Portal: The attributes of the new FAIS_Portal. On input, these shall be initialized by the FAIS_Client, as described by 7.3.7, with the following exceptions:

a) **RegionId:** On input, the FAIS_Client shall initialize this argument to the identifier of a specific FAIS_Region with which to associate the FAIS_Portal being created, or to FAIS_REGION_ID_ANY to indicate that the FAIS_Provider shall select the FAIS_Region. On

output, the FAIS_Provider shall initialize this argument to the identifier of the FAIS_Region that was assigned to the created FAIS_Portal. If the FAIS_Provider was unable to assign a FAIS_Region, the operation shall fail returning the appropriate FAIS_STATUS_T (see 6.3.9); and

b) **Portal_id:** On input, the FAIS_Client shall initialize this argument to an identifier for the FAIS_Portal being created or to FAIS_PORTAL_ID_ANY to indicate that the FAIS_Provider shall select the identifier. On output, the FAIS_Provider shall initialize this argument to the identifier assigned to the created FAIS_Portal. If the FAIS_Provider was unable to assign an identifier, the operation shall fail returning the appropriate FAIS_STATUS_T (see 6.3.9).

filter: Filter for enumerating both FTs and BIs.

nFTs: The number of FTs. On input, the FAIS_Client specifies the maximum number of FTs for which it is able to get status (i.e., as limited by the buffer referenced by pFTs). If set to 0 on input, no FTs are returned and the pFTs argument is ignored. On output, the FAIS_Provider updates this to indicate the number of FTs actually found.

pFTs: A pointer to a buffer for storing the details of the FTs that were found. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nFTs. This argument is optional, and is ignored if the nFTs argument is 0.

nBIs: The number of BIs. On input, the FAIS_Client specifies the maximum number of BIs for which it is able to get status (i.e., as limited by the buffer referenced by pBIs). If set to 0 on input, no BIs are returned and the pBIs argument is ignored. On output, the FAIS_Provider updates this to indicate the number of BIs actually found.

pBIs: A pointer to a buffer for storing the details of the BIs that were found. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nBIs. This argument is optional, and is ignored if the nBIs argument is 0.

7.4.7.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8 Front-end services

8.1 Overview

Front-end Services provide support for processing requests that are received from a remote SCSI Initiator.

8.2 Data structures

8.2.1 FAIS_FT

8.2.1.1 Format

```
typedef struct FAIS_FT_S {
        FAIS_ObjectCHandle_T ObjCHandle;
        FAIS_PortName_T Name;
} FAIS_FT_T;
```

8.2.1.2 Description

Data structure for an FT. See also 5.4.13.

8.2.1.3 Members

ObjCHandle: Handle assigned by the FAIS_Client to this object.

Name: Name of this FT.

8.2.2 FAIS_FIT

8.2.2.1 Format

```
typedef struct FAIS_FIT_S {
        FAIS_ObjectCHandle_T ObjCHandle;
        FAIS_ObjectPHandle_T FTObjPHandle;
        FAIS_PortName_T InitiatorName;
} FAIS_FIT_T;
```

8.2.2.2 Description

Data structure for an FIT. See also 5.4.10.

8.2.2.3 Members

ObjCHandle: Handle assigned by the FAIS_Client to this object.

FTObjPHandle: Handle assigned by the FAIS_Provider to the associated FT.

InitiatorName: Name of the initiator remote SCSI port.

8.2.3 FAIS_FITL

8.2.3.1 Format

```
typedef struct FAIS_FITL_S {
        FAIS_ObjectCHandle_T ObjCHandle;
        FAIS_ObjectPHandle_T FITObjPHandle;
        FAIS_LUN_T LUN;
        FAIS_ObjectPHandle_T VDEVObjPHandle;
```

```
        FAIS_FITLPermission_T ReadPermission;
        FAIS_FITLPermission_T WritePermission;
} FAIS_FITL_T;
```

8.2.3.2 Description

Data structure for a FITL. See also 5.4.11.

8.2.3.3 Members

ObjCHandle: Handle assigned by the FAIS_Client to this object.

FITObjPHandle: Handle assigned by the FAIS_Provider to the associated FIT.

LUN: Logical unit number of this FITL.

FLUObjPHandle: Handle assigned by the FAIS_Provider to the associated FLUVDEV.

ReadPermission: Permission to be applied for SCSI read commands to this FITL.

WritePermission: Permission to be applied for SCSI write commands to this FITL.

8.2.4 FAIS_FITLPermission

8.2.4.1 Format

```
typedef enum FAIS_FITLPermission_E {
        FAIS_FITL_ALLOW = 1,
        FAIS_FITL_FAULT_ONE = 2,
        FAIS_FITL_FAULT_ALL = 3,
        FAIS_FITL_HOLD = 4,
        FAIS_FITL_CONTROL = 5,
} FAIS_FITLPermission_T;
```

8.2.4.2 Description

The enumeration FAIS_FITLPermission_E defines the types of actions to take on FITL.

There is an additional requirement of Draining/Completing all outstanding IOs before a particular FITL Permission is applied. All the I/O errors due to draining/completion of I/Os should be delivered before the completion of the fais_FITL_UpdatePermission calls.

8.2.4.3 Values

FAIS_FITL_ALLOW: Allow the offload of held and new I/Os to proceed.

FAIS_FITL_FAULT_ONE: Fault all new I/Os on this FITL, but only generate an event notification on the first one.

FAIS_FITL_FAULT_ALL: Fault all future IOs on this FITL.

FAIS_FITL_HOLD: Hold all future IOs on this FITL.

FAIS_FITL_CONTROL: Forward all I/Os on this FITL to the FAIS_Client.

8.3 Function Calls

8.3.1 fais_FT_Create

8.3.1.1 Format

```
typedef struct FAIS_FT_Create_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectPHandle_T ObjPHandle;
    IN  uint32 flags;
    IN  FAIS_FT_T FT;
} FAIS_FT_Create_T;

FAIS_Status_T fais_FT_Create (FAIS_FT_Create_T * ptr);
```

8.3.1.2 Description

Creates an FT.

8.3.1.3 Arguments

ptr: A pointer to a FAIS_FT_Create_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

flags: A bit field with zero or more of the flags as shown in table 11.

Table 11 – fais_FT_Create flags

Bit(s)	Flag	Description
0	FAIS_PORT_NAME_AUTO	The name of the new FT shall be assigned automatically.
1 - 31	Reserved	

FT: On input this shall be initialized by the FAIS_Client, as described by, as described by 8.2.1, with the following exceptions:

a) **Name:** On input, the FAIS_Client shall initialize this argument to the name for the FT being created, if FAIS_PORT_NAME_AUTO is not set; otherwise, this is ignored on input. On output, the FAIS_Provider shall initialize this argument to the name assigned to the created FT. If the FAIS_Provider was unable to assign a name, the operation shall fail with the appropriate FAIS_STATUS_T (see 6.3.9).

8.3.1.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.2 fais_FT_Destroy

8.3.2.1 Format

```
typedef struct FAIS_FT_Destroy_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
```

```
} FAIS_FT_Destroy_T;

FAIS_Status_T fais_FT_Destroy (FAIS_FT_Destroy_T * ptr);
```

8.3.2.2 Description

Destroys a SCSI Front-end Target (FT). Destroying the FT implicitly deactivates any FAIS_Portals with which it is associated.

8.3.2.3 Arguments

ptr: A pointer to a FAIS_FT_Destroy_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

8.3.2.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.3 fais_FT_Activate

8.3.3.1 Format

```
typedef struct FAIS_FT_Activate_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_ObjectPHandle_T PortalObjPHandle;
} FAIS_FT_Activate_T;

FAIS_Status_T fais_FT_Activate (FAIS_FT_Activate_T * ptr);
```

8.3.3.2 Description

Associates an FT with a FAIS_Portal. Once activated, the FT is now visible to Remote SCSI Ports via the specified FAIS_Portal.

When iSCSI is used as the transport protocol, the fais_FT_Activate function may be called more than once for an FT if there are multiple portals associated with the FT. When FCP is used as the transport, the fais_FT_Active shall be called only once for an FT. If it is called more than once for FCP, the additional call shall fail with error code FAIS_STATUS_NO_SPACE.

8.3.3.3 Arguments

ptr: A pointer to a FAIS_FT_Activate_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this FT object assigned by the FAIS_Client.

ObjPHandle: The handle for this FT object assigned by the FAIS_Provider.

PortalObjPHandle: The FAIS_Portal object handle, assigned by the FAIS_Provider, on which FT is to be activated.

8.3.3.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.4 fais_FT_Deactivate

8.3.4.1 Format

```
typedef struct FAIS_FT_Deactivate_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_ObjectPHandle_T PortalObjPHandle;
} FAIS_FT_Deactivate_T;

FAIS_Status_T fais_FT_Deactivate (FAIS_FT_Deactivate_T *ptr);
```

8.3.4.2 Description

Disassociates an FT from a FAIS_Portal. Once deactivated, the FT is no longer visible to Remote SCSI Ports via the specified FAIS_Portal.

8.3.4.3 Arguments

ptr: A pointer to a FAIS_FT_Deactivate_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this FT object assigned by the FAIS_Client.

ObjPHandle: The handle for this FT object assigned by the FAIS_Provider.

PortalObjPHandle: The FAIS_Portal object handle, assigned by the FAIS_Provider, on which FT is to be deactivated.

8.3.4.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.5 fais_FT_GetStatus

8.3.5.1 Format

```
typedef struct FAIS_FT_GetStatus_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    OUT FAIS_FT_T FT;
    IN  FAIS_EnumerationFilter_T filter;
    INOUTuint32 nPortals;
    OUT FAIS_HandleSet_T *pPortals;
    INOUTuint32 nFITs;
    OUT FAIS_HandleSet_T *pFITs;
} FAIS_FT_GetStatus_T;
```

```
FAIS_Status_T fais_FT_GetStatus (FAIS_FT_GetStatus_T * ptr);
```

8.3.5.2 Description

Gets the status of an FT.

8.3.5.3 Arguments

ptr: A pointer to a FAIS_FT_GetStatus_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

FT: The attributes of the requested FT; see 8.2.1 for a detailed description.

filter: Filter for enumerating both Portals and FITs.

nPortals: The number of FAIS_Portals. On input, the FAIS_Client specifies the maximum number of FAIS_Portals it is able to get status for (i.e., as limited by the buffer referenced by pPortals). If set to 0 on input, no FAIS_Portals are returned and the pPortals argument is ignored. On output, the FAIS_Provider updates this parameter to indicate the number of FAIS_Portals actually found; this may be greater than the number of entries actually populated in the pPortals buffer.

pPortals: A pointer to a buffer for storing the identifiers of FAIS Portals, where FT was activated. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nPortals. This argument is optional, and is ignored if the nPortals argument is 0.

nFITs: The number of FITs. On input, the FAIS_Client specifies the maximum number of FITs for which it is able to get status (i.e., as limited by the buffer referenced by pFITs). If set to 0 on input, no FITs are returned and the pFITs argument is ignored. On output, the FAIS_Provider updates this to indicate the number of FITs actually found.

pFITs: A pointer to a buffer for storing the identifiers of FITs associated with the FT. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nFITs. This argument is optional, and is ignored if the nFITs argument is 0.

8.3.5.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.6 fais_FIT_Create

8.3.6.1 Format

```
typedef struct FAIS_FIT_Create_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_FIT_T FIT;
} FAIS_FIT_Create_T;

FAIS_Status_T fais_FIT_Create (FAIS_FIT_Create_T * ptr);
```

8.3.6.2 Description

Creates a FIT.

8.3.6.3 Arguments

ptr: A pointer to a FAIS_FIT_Create_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

FIT: The attributes of the new FIT; see 8.2.2 for detailed description.

8.3.6.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.7 fais_FIT_Destroy

8.3.7.1 Format

```
typedef struct FAIS_FIT_Destroy_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
} FAIS_FIT_Destroy_T;

FAIS_Status_T fais_FIT_Destroy (FAIS_FIT_Destroy_T * ptr);
```

8.3.7.2 Description

Destroys a Front-end Initiator Target (FIT) object.

8.3.7.3 Arguments

ptr: A pointer to a FAIS_FIT_Destroy_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

8.3.7.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.8 fais_FIT_GetStatus

8.3.8.1 Format

```
typedef struct FAIS_FIT_GetStatus_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    OUT FAIS_FIT_T FIT;
    IN  FAIS_EnumerationFilter_T filter;
    INOUTuint32 nFITLs;
    OUT FAIS_HandleSet_T *pFITLs;
```

```
} FAIS_FIT_GetStatus_T;

FAIS_Status_T fais_FIT_GetStatus (FAIS_FIT_GetStatus_T * ptr);
```

8.3.8.2 Description

Gets the status of a FIT.

8.3.8.3 Arguments

ptr: A pointer to a FAIS_FIT_GetStatus_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

FIT: The attributes of the requested FIT; see 8.2.2 for a detailed description.

filter: Filter for enumerating FITLs.

nFITLs: The number of FITLs. On input, the FAIS_Client specifies the maximum number of FITLs for which it is able to get status (i.e., as limited by the buffer referenced by pFITLs). If set to 0 on input, no FITLs are returned and the pFITLs argument is ignored. On output, the FAIS_Provider updates this to indicate the number of FITLs actually found.

pFITLs: A pointer to a buffer for storing the identifiers of FITLs associated with the FIT. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nFITLs. This argument is optional, and is ignored if the nFITLs argument is 0.

8.3.8.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.9 fais_FITL_Create

8.3.9.1 Format

```
typedef struct FAIS_FITL_Create_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_FITL_T FITL;
} FAIS_FITL_Create_T;

FAIS_Status_T fais_FITL_Create (FAIS_FITL_Create_T * ptr);
```

8.3.9.2 Description

Creates a FITL.

8.3.9.3 Arguments

ptr: A pointer to a FAIS_FITL_Create_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

FITL: The attributes of the new FITL; see 8.2.3 for a detailed description.

8.3.9.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.10 fais_FITL_Destroy

8.3.10.1 Format

```
typedef struct FAIS_FITL_Destroy_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
} FAIS_FITL_Destroy_T;

FAIS_Status_T fais_FITL_Destroy (FAIS_FITL_Destroy_T * ptr);
```

8.3.10.2 Description

This method destroys a FITL object and frees any resources that were allocated during the fais_FITL_Create.

8.3.10.3 Arguments

ptr: A pointer to a FAIS_FITL_Destroy_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

8.3.10.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.11 fais_FITL_UpdatePermission

8.3.11.1 Format

```
typedef struct FAIS_FITL_UpdatePermission_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_FITLPermission_T ReadPermission;
    IN  FAIS_FITLPermission_T WritePermission;
} FAIS_FITL_UpdatePermission_T;

FAIS_Status_T fais_FITL_UpdatePermission (FAIS_FITL_UpdatePermission_T *
ptr);
```

8.3.11.2 Description

Updates Permission of the FITL.

8.3.11.3 Arguments

ptr: A pointer to a FAIS_FITL_UpdatePermission_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

ReadPermission: Permission to be applied for READs on the FITL.

WritePermission: Permission to be applied for WRITEs on the FITL.

8.3.11.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.12 fais_FITL_AbortIOs

8.3.12.1 Format

```
typedef struct FAIS_FITL_AbortIOs_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
} FAIS_FITL_AbortIOs_T;

FAIS_Status_T fais_FITL_AbortIOs (FAIS_FITL_AbortIOs_T * ptr);
```

8.3.12.2 Description

Abort all the outstanding IOs on this FITL.

8.3.12.3 Arguments

ptr: A pointer to a FAIS_FITL_AbortIOs_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

8.3.12.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

8.3.13 fais_FITL_GetStatus

8.3.13.1 Format

```
typedef struct FAIS_FITL_GetStatus_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
```

```
    OUT FAIS_FITL_T FITL;
} FAIS_FITL_GetStatus_T;

FAIS_Status_T fais_FITL_GetStatus (FAIS_FITL_GetStatus_T * ptr);
```

8.3.13.2 Description

Gets the status of a FITL.

8.3.13.3 Arguments

ptr: A pointer to a FAIS_FITL_GetStatus_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

FITL: The attributes of the requested FITL; see 8.2.3 for a detailed description.

8.3.13.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9 Back-end services

9.1 Overview

Back-end Services provide support for discovering elements connected to the logical back-end of the FAIS_Platform and for sending commands to devices on the back-end.

9.2 Data structures

9.2.1 FAIS_BI

9.2.1.1 Format

```
typedef struct FAIS_BI_S {
        FAIS_ObjectCHandle_T ObjCHandle;
        FAIS_PortName_T Name;
} FAIS_BI_T;
```

9.2.1.2 Description

Data structure for a BI. See also 5.4.1.

9.2.1.3 Members

ObjCHandle: Handle assigned by the FAIS_Client to this object.

Name: Name for this BI.

9.2.2 FAIS_BIT

9.2.2.1 Format

```
typedef struct FAIS_BIT_S {
        FAIS_ObjectCHandle_T ObjCHandle;
        FAIS_ObjectPHandle_T BIObjPHandle;
        FAIS_PortName_T TargetName;
} FAIS_BIT_T;
```

9.2.2.2 Description

Data structure for an BIT. See also 5.4.2.

9.2.2.3 Members

ObjCHandle: Handle assigned by the FAIS_Client to this object.

BIObjPHandle: Handle assigned by the FAIS_Provider to the associated BI.

TargetName: Name of the target remote SCSI port.

9.2.3 FAIS_SCSI_CDB

9.2.3.1 Format

```
typedef uint8 FAIS_SCSI_CDB[FAIS_SCSI_CDBLEN_MAX];

typedef FAIS_SCSI_CDB FAIS_SCSI_CDB_T;
```

9.2.3.2 Description

Data structure defining a SCSI command description block. This data structure defines the SCSI Command Description Block (CDB) for use in the FAIS_BI_SendSCSICommand .

9.2.4 FAIS_SCSIFlag

9.2.4.1 Format

```
typedef enum FAIS_SCSIFlag_E {
        FAIS_SCSI_FLAG_NODATA = 1,
        FAIS_SCSI_FLAG_DATA_IN = 2,
        FAIS_SCSI_FLAG_DATA_OUT = 3,
} FAIS_SCSIFlag_T;
```

9.2.4.2 Description

The enumeration FAIS_SCSIFlag_E defines the flag to identify the type of SCSI command. These codes are used for sending and receiving SCSI commands.

9.2.4.3 Values

FAIS_SCSI_FLAG_NODATA: No data transfer.

FAIS_SCSI_FLAG_DATA_IN: Data transfer from Target to Initiator.

FAIS_SCSI_FLAG_DATA_OUT: Data transfer from Initiator to Target.

NOTE SCSI Bidirectional commands are not supported by this standard.

9.2.5 FAIS_SCSIStatus

9.2.5.1 Format

```
typedef uint32 FAIS_SCSIStatus_T;
```

9.2.5.2 Description

FAIS_SCSIStatus_T defines the return status codes for a SCSI command.

The values for the status codes shall be as defined in SAM-3.

9.2.6 FAIS_ResidualFlag

9.2.6.1 Format

```
typedef enum FAIS_ResidualFlag_E {
        FAIS_RESIDUAL_NONE = 0,
        FAIS_RESIDUAL_UNDERRUN = 1,
        FAIS_RESIDUAL_OVERRUN = 2,
} FAIS_ResidualFlag_T;
```

9.2.6.2 Description

Defines whether an underrun or overrun condition occurred during SCSI command processing.

9.2.6.3 Members

FAIS_RESIDUAL_NONE: An underrun/overrun condition did not occur.

FAIS_RESIDUAL_UNDERRUN: An underrun condition occurred.

FAIS_RESIDUAL_OVERRUN: An overrun condition occurred.

9.2.7 FAIS_TaskMgmtCmd

9.2.7.1 Format

```
typedef enum FAIS_TaskMgmtCmd_E {
        FAIS_TASK_MGMT_CMD_NONE = 0,
        FAIS_TASK_MGMT_CMD_ABORT_TASK_SET = 1,
        FAIS_TASK_MGMT_CMD_CLEAR_TASK_SET = 2,
        FAIS_TASK_MGMT_CMD_LUN_RESET = 3,
        FAIS_TASK_MGMT_CMD_TARGET_RESET = 4,
        FAIS_TASK_MGMT_CMD_CLEAR_ACA = 5
} FAIS_TaskMgmtCmd_T;
```

9.2.7.2 Description

Task Management commands supported by FAIS.

9.2.7.3 Values

FAIS_TASK_MGMT_NONE: No task management function is to be performed.

FAIS_TASK_MGMT_ABORT_TASK_SET: Abort all commands in a task set (see SAM-3).

FAIS_TASK_MGMT_CLEAR_TASK_SET: Clear all commands in a task set (see SAM-3).

FAIS_TASK_MGMT_LUN_RESET: Reset a LUN (see SAM-3).

FAIS_TASK_MGMT_TARGET_RESET: Reset all LUNs on a Target (see SAM-3).

FAIS_TASK_MGMT_CLEAR_ACA: Clear Automatic Contingent Allegiance condition (see SAM-3).

9.2.8 FAIS_TaskAttribute

9.2.8.1 Format

```
typedef enum FAIS_TaskAttribute_E {
        FAIS_TASK_ATTRIBUTE_SIMPLE = 0,
        FAIS_TASK_ATTRIBUTE_HEAD_OF_QUEUE = 1,
        FAIS_TASK_ATTRIBUTE_ORDERED = 2,
        FAIS_TASK_ATTRIBUTE_UNTAGGED = 3,
        FAIS_TASK_ATTRIBUTE_ACA = 4,
} FAIS_TaskAttribute_T;
```

9.2.8.2 Description

Defines the task attribute (see SAM-3) associated with the CDB.

9.2.8.3 Values

FAIS_TASK_ATTRIBUTE_SIMPLE: The task shall be managed according to the rules for a simple task attribute (see SAM-3).

FAIS_TASK_ATTRIBUTE_HEAD_OF_QUEUE: The task shall be managed according to the rules for a head of queue task attribute (see SAM-3).

FAIS_TASK_ATTRIBUTE_ORDERED: The task shall be managed according to the rules for an ordered task attribute (see SAM-3).

FAIS_TASK_ATTRIBUTE_UNTAGGED: Obsoleted in SAM-3.

FAIS_TASK_ATTRIBUTE_ACA: The task shall be managed according to the rules for an automatic contingent allegiance task attribute (see SAM-3).

9.2.9 FAIS_TaskMgmtResponse

9.2.9.1 Format

```
typedef enum FAIS_TaskMgmtResponse_E {
        FAIS_TASK_MGMT_RESPONSE_COMPLETE = 0,
        FAIS_TASK_MGMT_RESPONSE_REJECTED = 4,
        FAIS_TASK_MGMT_RESPONSE_FAILED = 5,
        FAIS_TASK_MGMT_RESPONSE_BAD_LUN = 9,
} FAIS_TaskMgmtResponse_T;
```

9.2.9.2 Description

Defines the SCSI transport protocol specific responses that shall be returned for a Task Management function (see SAM-3).

9.2.9.3 Values

FAIS_TASK_MGMT_RESPONSE_COMPLETE: The Task Management function completed (see SAM-3).

FAIS_TASK_MGMT_RESPONSE_REJECTED: The Task Management function was rejected (see SAM-3).

FAIS_TASK_MGMT_RESPONSE_FAILED: The Task Management function failed (see SAM-3).

FAIS_TASK_MGMT_RESPONSE_BAD_LUN: Incorrect LUN specified in the Task Management command (see SAM-3).

9.2.10 FAIS_BITL

9.2.10.1 Format

```
typedef struct FAIS_BITL_S {
        FAIS_ObjectCHandle_T ObjCHandle;
        FAIS_ObjectPHandle_T BITObjPHandle;
        FAIS_LUN_T LUN;
} FAIS_BITL_T;
```

9.2.10.2 Description

Data structure defining a BITL. See also 5.4.3.

9.2.10.3 Members

ObjCHandle: Handle assigned by the FAIS_Client to this object.

BITObjPHandle: FAIS_Provider-assigned handle of the associated BIT.

LUN: Logical unit number of this BITL.

9.2.11 FAIS_SCSI_SCB

9.2.11.1 Format

```
typedef struct FAIS_SCB_S {
        FAIS_TaskMgmtCmd_T TaskMgmtCmd;
        uint8 CdbLen;
        FAIS_SCSI_CDB_T Cdb;
        FAIS_TaskAttribute_T TaskAttribute;
        uint8 TaskPriority;
        FAIS_SCSIFlag_T SCSIFlag;
        FAIS_TaskMgmtResponse_T TaskMgmtResponse;
        FAIS_SCSIStatus_T SCSIStatus;
        FAIS_ResidualFlag_T ResidualFlag;
        uint32 ResidualLen;
        uint32 SenseDataLen;
        uint8 *SenseData;
        uint32 DataLen;
        uint8 *Data;
} FAIS_SCB_T;
```

9.2.11.2 Description

Data structure defining a SCSI control block. This data structure defines the values needed to send a SCSI command to a backend device. The creator of the SCB structure is required to allocate and free memory for SenseData and Data fields.

9.2.11.3 Members

TaskMgmtCmd: The task management command to be performed. If this field is set to any value other than FAIS_TASK_MGMT_CMD_NONE, the CdbLen, Cdb, TaskAttribute, TaskPriority, and SCSIFlag fields are ignored on input and SCSIStatus, ResidualFlag, ResidualLen, SenseDataLen, SenseData, DataLen, Data are invalid on return. If this field is set to FAIS_TASK_MGMT_CMD_NONE, TaskMgmtResponse is invalid on return.

CdbLen: Length of the CDB in bytes.

Cdb: CDB.

TaskAttribute: SCSI command task attribute (see 9.2.8).

TaskPriority: Specifies the relative scheduling of this task in relation to other tasks already in the task set (see SAM-3). If the TaskAttribute field contains a value other than SIMPLE, then this field is reserved.

SCSIFlag: Specifies the direction of data flow for the SCSI command defined in the CDB (see 9.2.4).

TaskMgmtResponse: The completion status of a task management command (see 9.2.9). This field is set on return for all TaskMgmtCmd other than FAIS_TASK_MGMT_CMD_NONE.

SCSIStatus: The completion status of a SCSI command (see 9.2.5).

ResidualFlag: Specifies a residual underrun/overrun condition. Set on return of a SCSI command (see 9.2.6).

ResidualLen: Number of bytes underrun or overrun. Set on return of a SCSI command if an underrun or overrun condition occurred.

SenseDataLen: Length of the SenseData field in bytes.

SenseData: Pointer to the SenseData buffer (see SPC-3). This needs to be preallocated by the caller.

DataLen: Length of the Data field in bytes.

Data: Pointer to the read/write data buffer.

9.3 Function Calls

9.3.1 fais_BI_Create

9.3.1.1 Format

```
typedef struct FAIS_BI_Create_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectPHandle_T ObjPHandle;
    IN  uint32 flags;
    IN  FAIS_BI_T BI;
} FAIS_BI_Create_T;

FAIS_Status_T fais_BI_Create (FAIS_BI_Create_T * ptr);
```

9.3.1.2 Description

Creates a BI.

9.3.1.3 Arguments

ptr: A pointer to a FAIS_BI_Create_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

flags: A bit field with zero or more of the flags as shown in table 12.

Table 12 – fais_BI_Create flags

Bit(s)	Flag	Description
0	FAIS_PORT_NAME_AUTO	The name of the new BI shall be assigned automatically.
1 to 31	Reserved	

BI: On input this shall be initialized by the FAIS_Client, as described by, as described by 9.2.1, with the following exception:

a) **Name:** On input, the FAIS_Client shall initialize this argument to the name for the BI being created, if FAIS_PORT_NAME_AUTO is not set; otherwise, this is ignored on input. On output, the FAIS_Provider shall initialize this argument to the name assigned to the created BI. If the FAIS_Provider was unable to assign a name, the operation shall fail with the appropriate FAIS_STATUS_T (see 6.3.9).

9.3.1.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.2 fais_BI_Destroy

9.3.2.1 Format

```
typedef struct FAIS_BI_Destroy_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
} FAIS_BI_Destroy_T;

FAIS_Status_T fais_BI_Destroy (FAIS_BI_Destroy_T * ptr);
```

9.3.2.2 Description

Removes a SCSI Back-end Initiator (BI). Destroying the BI implicitly deactivates any FAIS_Portals with which it is associated.

9.3.2.3 Arguments

ptr: A pointer to a FAIS_BI_Destroy_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

9.3.2.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.3 fais_BI_Activate

9.3.3.1 Format

```
typedef struct FAIS_BI_Activate_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_ObjectPHandle_T PortalObjPHandle;
} FAIS_BI_Activate_T;

FAIS_Status_T fais_BI_Activate (FAIS_BI_Activate_T * ptr);
```

9.3.3.2 Description

Associates a BI with a FAIS_Portal. Once activated, the BI is visible to Remote SCSI Ports via the specified FAIS_Portal.

9.3.3.3 Arguments

ptr: A pointer to a FAIS_BI_Activate_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this BI object assigned by the FAIS_Client.

ObjPHandle: The handle for this BI object assigned by the FAIS_Provider.

PortalObjPHandle: The FAIS_Portal object handle, assigned by the FAIS_Provider, on which BI is to be activated.

9.3.3.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.4 fais_BI_Deactivate

9.3.4.1 Format

```
typedef struct FAIS_BI_Deactivate_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_ObjectPHandle_T PortalObjPHandle;
} FAIS_BI_Deactivate_T;

FAIS_Status_T fais_BI_Deactivate (FAIS_BI_Deactivate_T * ptr);
```

9.3.4.2 Description

Disassociates a BI from a FAIS_Portal. Once deactivated, the BI is no longer visible to Remote SCSI Ports via the specified FAIS_Portal.

9.3.4.3 Arguments

ptr: A pointer to a FAIS_BI_Deactivate_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this BI object assigned by the FAIS_Client.

ObjPHandle: The handle for this BI object assigned by the FAIS_Provider.

PortalObjPHandle: The FAIS_Portal object handle, assigned by the FAIS_Provider, on which BI is to be deactivated.

9.3.4.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.5 fais_BI_SendSCSICommand

9.3.5.1 Format

```
typedef struct FAIS_BI_SendSCSICommand_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_PortName_T TargetName;
    IN  FAIS_LUN_T LUN;
    INOUTFAIS_SCB_T * pScb;
} FAIS_BI_SendSCSICommand_T;
```

```
FAIS_Status_T fais_BI_SendSCSICommand (FAIS_BI_SendSCSICommand_T *);
```

9.3.5.2 Description

This method is used to send SCSI commands to a back-end device associated with a BI without creating a BIT and a BITL.

9.3.5.3 Arguments

ptr: A pointer to a FAIS_BI_SendSCSICommand_T structure, the fields of which fields are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

TargetName: The FAIS_SCSI_Port Name identifying the Back-end Target.

LUN: The LUN to which the command is sent.

pScb: A pointer to an SCB data structure

9.3.5.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.6 fais_BI_GetStatus

9.3.6.1 Format

```
typedef struct FAIS_BI_GetStatus_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    OUT FAIS_BI_T BI;
    IN  FAIS_EnumerationFilter_T filter;
    INOUTuint32 nPortals;
    OUT FAIS_HandleSet_T *pPortals;
    INOUTuint32 nBITs;
    OUT FAIS_HandleSet_T *pBITs;
} FAIS_BI_GetStatus_T;

FAIS_Status_T fais_BI_GetStatus (FAIS_BI_GetStatus_T * ptr);
```

9.3.6.2 Description

Gets the status of a BI.

9.3.6.3 Arguments

ptr: A pointer to a FAIS_BI_GetStatus_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

BI: The attributes of the requested BI; see 9.2.1 for a detailed description.

filter: Filter for enumerating both Portals and BITs.

nPortals: The number of FAIS Portals.

pPortals: A pointer to a buffer for storing the identifiers of FAIS Portals, where BI was activated. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nPortals. This argument is optional, and is ignored if the nPortals argument is 0.

nBITs: The number of BITs. On input, the FAIS_Client specifies the maximum number of BITs for which it is able to get status (i.e., as limited by the buffer referenced by pBITs). If set to 0 on input, no BITs are returned and the pBITs argument is ignored. On output, the FAIS_Provider updates this to indicate the number of BITs actually found.

pBITs: A pointer to a buffer for storing the identifiers of BITs associated with the BT. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nBITs. This argument is optional, and is ignored if the nBITs argument is 0.

9.3.6.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.7 fais_BIT_Create

9.3.7.1 Format

```
typedef struct FAIS_BIT_Create_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_BIT_T BIT;
} FAIS_BIT_Create_T;

FAIS_Status_T fais_BIT_Create (FAIS_BIT_Create_T * ptr);
```

9.3.7.2 Description

Creates a BIT.

9.3.7.3 Arguments

ptr: A pointer to a FAIS_BIT_Create_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

BIT: The attributes of the new BIT; see 9.2.2 for a detailed description.

9.3.7.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.8 fais_BIT_Destroy

9.3.8.1 Format

```
typedef struct FAIS_BIT_Destroy_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
} FAIS_BIT_Destroy_T;

FAIS_Status_T fais_BIT_Destroy (FAIS_BIT_Destroy_T * ptr);
```

9.3.8.2 Description

Destroys a Back-end Initiator Target (BIT) object.

9.3.8.3 Arguments

ptr: A pointer to a FAIS_BIT_Destroy_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

9.3.8.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.9 fais_BIT_SendSCSICommand

9.3.9.1 Format

```
typedef struct FAIS_BIT_SendSCSICommand_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_LUN_T LUN;
    INOUTFAIS_SCB_T * pScb;
} FAIS_BIT_SendSCSICommand_T;

FAIS_Status_T fais_BIT_SendSCSICommand (FAIS_BIT_SendSCSICommand_T *);
```

9.3.9.2 Description

This method is used to send SCSI commands to a back-end device associated with a BIT without creating a BITL.

9.3.9.3 Arguments

ptr: A pointer to a FAIS_BIT_SendSCSICommand_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

LUN: The LUN to which the command is sent.

pScb: A pointer to an SCB data structure

9.3.9.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.10 fais_BIT_GetStatus

9.3.10.1 Format

```
typedef struct FAIS_BIT_GetStatus_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
    OUT FAIS_BIT_T BIT;
    IN  FAIS_EnumerationFilter_T filter;
    INOUTuint32 nBITLs;
    OUT FAIS_HandleSet_T *pBITLs;
} FAIS_BIT_GetStatus_T;

FAIS_Status_T fais_BIT_GetStatus (FAIS_BIT_GetStatus_T * ptr);
```

9.3.10.2 Description

Gets the status of a BIT.

9.3.10.3 Arguments

ptr: A pointer to a FAIS_BIT_GetStatus_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

BIT: The attributes of the requested BIT; see 9.2.2 for a detailed description.

filter: Filter for enumerating BITLs.

nBITLs: The number of BITLs. On input, the FAIS_Client specifies the maximum number of BITLs for which it is able to get status (i.e., as limited by the buffer referenced by pBITLs). If set to 0 on input, no BITLs are returned and the pBITLs argument is ignored. On output, the FAIS_Provider updates this to indicate the number of BITLs actually found.

pBITLs: A pointer to a buffer for storing the identifiers of BITLs associated with the BIT. The buffer shall be large enough to support up to the number specified by the FAIS_Client in nBITLs. This argument is optional, and is ignored if the nBITLs argument is 0.

9.3.10.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.11 fais_BITL_Create

9.3.11.1 Format

```
typedef struct FAIS_BITL_Create_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectPHandle_T ObjPHandle;
    IN  FAIS_BITL_T BITL;
} FAIS_BITL_Create_T;

FAIS_Status_T fais_BITL_Create (FAIS_BITL_Create_T * ptr);
```

9.3.11.2 Description

Creates a BITL.

9.3.11.3 Arguments

ptr: A pointer to a FAIS_BITL_Create_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjPHandle: The handle for this object assigned by the FAIS_Provider.

BITL: The attributes of the new BITL; see 9.2.10 for a detailed description.

9.3.11.4 Return values

FAIS_STATUS_SUCCESS upon successful acceptance of the asynchronous call. Otherwise, returns the status condition identifying the error.

9.3.12 fais_BITL_Destroy

9.3.12.1 Format

```
typedef struct FAIS_BITL_Destroy_S {
    INOUTFAIS_ClientRequest_Header_T RequestHeader;
    OUT FAIS_ObjectCHandle_T ObjCHandle;
    IN  FAIS_ObjectPHandle_T ObjPHandle;
} FAIS_BITL_Destroy_T;

FAIS_Status_T fais_BITL_Destroy (FAIS_BITL_Destroy_T * ptr);
```

9.3.12.2 Description

This method removes a BITL object and frees any resources that were allocated during the fais_BITL_Create.

9.3.12.3 Arguments

ptr: A pointer to a FAIS_BITL_Destroy_T structure, the fields of which are defined in this subclause as arguments.

RequestHeader: The common parameters for asynchronous-capable request methods (see 6.3.2).

ObjCHandle: The handle for this object assigned by the FAIS_Client.

ObjPHandle: The handle for this object assigned by the FAIS_Provider.