



ISO/IEC 11989

Edition 1.0 2010-03

INTERNATIONAL STANDARD

Information technology – iSCSI management API



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2010 ISO/IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester.

If you have any questions about ISO/IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

IEC Central Office
3, rue de Varembe
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 It also gives information on projects, withdrawn and replaced publications.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 11989

Edition 1.0 2010-03

INTERNATIONAL STANDARD

Information technology – iSCSI management API

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

PRICE CODE **XD**

ICS 35.200

ISBN 2-8318-1082-5

CONTENTS

FOREWORD.....	10
1 Scope.....	12
2 Normative references	12
3 Document conventions	12
4 Background technical information	14
4.1 Terms, definitions and abbreviations	14
4.2 Concepts.....	17
4.2.1 Library and plugins	17
4.2.2 Object ID	18
4.2.3 Object ID list.....	19
4.2.4 The Shared Node versus Non-shared Nodes	19
4.2.5 Logical HBA	20
4.2.6 Target OIDs and Logical Unit OIDs.....	20
4.2.7 Software Initiators versus Hardware Initiators	21
4.2.8 iSCSI session and connection parameters.....	22
4.2.9 Class relationship diagram	23
5 Constants and types.....	24
5.1 IMA_WCHAR.....	24
5.2 IMA_BYTE	24
5.3 IMA_BOOL.....	24
5.4 IMA_XBOOL.....	24
5.5 IMA_UINT	24
5.6 IMA_UINT16.....	24
5.7 IMA_UINT32.....	24
5.8 IMA_UINT64.....	24
5.9 IMA_DATETIME	24
5.10 IMA_OBJECT_VISIBILITY_FN	24
5.11 IMA_OBJECT_PROPERTY_FN.....	25
5.12 IMA_OBJECT_TYPE	25
5.12.1 Format.....	25
5.12.2 Fields	26
5.13 IMA_STATUS.....	27
5.13.1 Macros	27
5.13.2 Non-error statuses.....	27
5.13.3 Error statuses.....	28
5.14 IMA_OID	31
5.14.1 Format.....	31
5.14.2 Fields	31
5.14.3 Remarks.....	31
5.15 IMA_OID_LIST	32
5.15.1 Format.....	32
5.15.2 Fields	32
5.15.3 Remarks.....	32
5.16 IMA_NODE_NAME.....	32
5.16.1 Format.....	32
5.16.2 Remarks.....	32

5.17	IMA_NODE_ALIAS	32
5.17.1	Format.....	32
5.17.2	Remarks.....	32
5.18	IMA_IP_ADDRESS.....	33
5.18.1	Format.....	33
5.18.2	Fields	33
5.19	IMA_HOST_NAME	33
5.19.1	Format.....	33
5.19.2	Remarks.....	33
5.20	IMA_HOST_ID.....	33
5.20.1	Format.....	33
5.20.2	Fields	34
5.20.3	Remarks.....	34
5.21	IMA_TARGET_ADDRESS	34
5.21.1	Format.....	34
5.21.2	Fields	34
5.21.3	Remarks.....	34
5.22	IMA_ADDRESS_KEY	34
5.22.1	Format.....	34
5.22.2	Fields	35
5.22.3	Remarks.....	35
5.23	IMA_ADDRESS_KEYS	35
5.23.1	Format.....	35
5.23.2	Fields	35
5.23.3	Remarks.....	35
5.24	IMA_STATIC_DISCOVERY_TARGET.....	36
5.24.1	Format.....	36
5.24.2	Fields	36
5.24.3	Remarks.....	36
5.25	IMA_DISCOVERY_ADDRESS_PROPERTIES	36
5.25.1	Format.....	36
5.25.2	Fields	36
5.25.3	Remarks.....	37
5.26	IMA_STATIC_DISCOVERY_TARGET_PROPERTIES.....	37
5.26.1	Format.....	37
5.26.2	Fields	37
5.26.3	Remarks.....	37
5.27	IMA_IP_PROPERTIES	37
5.27.1	Format.....	37
5.27.2	Fields	38
5.27.3	Remarks.....	39
5.28	IMA_LIBRARY_PROPERTIES.....	39
5.28.1	Format.....	39
5.28.2	Fields	39
5.28.3	Remarks.....	40
5.29	IMA_PLUGIN_PROPERTIES.....	40
5.29.1	Format.....	40
5.29.2	Fields	40
5.29.3	Remarks.....	40

5.30	IMA_NODE_PROPERTIES	41
5.30.1	Format	41
5.30.2	Fields	41
5.30.3	Remarks	42
5.31	IMA_LHBA_PROPERTIES	42
5.31.1	Format	42
5.31.2	Fields	43
5.31.3	Remarks	43
5.32	Upper Level Protocol (ULP) flags	43
5.33	IMA_PHBA_PROPERTIES	44
5.33.1	Format	44
5.33.2	Fields	44
5.33.3	Remarks	46
5.34	IMA_DISCOVERY_PROPERTIES	47
5.34.1	Format	47
5.34.2	Fields	47
5.34.3	Remarks	48
5.35	IMA_PHBA_DOWNLOAD_IMAGE_TYPE	48
5.35.1	Format	48
5.35.2	Fields	48
5.35.3	Remarks	49
5.36	IMA_PHBA_DOWNLOAD_IMAGE_PROPERTIES	49
5.36.1	Format	49
5.36.2	Fields	49
5.36.3	Remarks	49
5.37	IMA_ISNS_DISCOVERY_METHOD	49
5.37.1	Format	49
5.37.2	Fields	50
5.37.3	Remarks	50
5.38	IMA_PHBA_DOWNLOAD_PROPERTIES	50
5.38.1	Format	50
5.38.2	Fields	50
5.38.3	Remarks	51
5.39	IMA_IPSEC_PROPERTIES	51
5.39.1	Format	51
5.39.2	Fields	51
5.39.3	Remarks	51
5.40	IMA_MIN_MAX_VALUE	51
5.40.1	Format	51
5.40.2	Fields	52
5.40.3	Remarks	52
5.41	IMA_BOOL_VALUE	53
5.41.1	Format	53
5.41.2	Fields	53
5.42	IMA_MAC_ADDRESS	53
5.42.1	Format	53
5.42.2	Remarks	53
5.43	IMA_LNP_PROPERTIES	53
5.43.1	Format	53

5.43.2	Fields	54
5.43.3	Remarks.....	54
5.44	IMA_PNP_PROPERTIES.....	54
5.44.1	Format.....	54
5.44.2	Fields	54
5.44.3	Remarks.....	55
5.45	IMA_PNP_STATISTICS.....	55
5.45.1	Format.....	55
5.45.2	Fields	55
5.45.3	Remarks.....	56
5.46	IMA_NETWORK_PORTAL_PROPERTIES.....	56
5.46.1	Format.....	56
5.46.2	Fields	56
5.46.3	Remarks.....	56
5.47	IMA_PHBA_STATUS.....	56
5.47.1	Format.....	56
5.47.2	Values	56
5.47.3	Remarks.....	56
5.48	IMA_NETWORK_PORT_STATUS	57
5.48.1	Format.....	57
5.48.2	Fields	57
5.48.3	Remarks.....	58
5.49	IMA_TARGET_DISCOVERY_METHOD	58
5.49.1	Format.....	58
5.49.2	Fields	58
5.50	IMA_TARGET_PROPERTIES.....	58
5.50.1	Format.....	58
5.50.2	Fields	59
5.50.3	Remarks.....	59
5.51	IMA_TARGET_ERROR_STATISTICS.....	60
5.51.1	Format.....	60
5.51.2	Fields	60
5.51.3	Remarks.....	61
5.52	IMA_LU_PROPERTIES	62
5.52.1	Format.....	62
5.52.2	Fields	62
5.52.3	Remarks.....	63
5.53	IMA_DEVICE_STATISTICS	63
5.53.1	Format.....	63
5.53.2	Fields	64
5.53.3	Remarks.....	64
5.54	IMA_STATISTICS_PROPERTIES.....	64
5.54.1	Format.....	64
5.54.2	Fields	64
5.54.3	Remarks.....	65
5.55	IMA_AUTHMETHOD	65
5.55.1	Format.....	65
5.55.2	Fields	65
5.56	IMA_CHAP_INITIATOR_AUTHPARMS	65

5.56.1	Format.....	66
5.56.2	Fields	66
5.56.3	Remarks.....	67
5.57	IMA_SRP_INITIATOR_AUTHPARMS	67
5.57.1	Format.....	67
5.57.2	Fields	67
5.57.3	Remarks.....	68
5.58	IMA_KRB5_INITIATOR_AUTHPARMS	68
5.58.1	Format.....	68
5.58.2	Fields	68
5.58.3	Remarks.....	68
5.59	IMA_SPKM_INITIATOR_AUTHPARMS.....	68
5.59.1	Format.....	68
5.59.2	Fields	69
5.59.3	Remarks.....	69
5.60	IMA_INITIATOR_AUTHPARMS	69
5.60.1	Format.....	69
5.60.2	Fields	70
5.60.3	Remarks.....	70
6	APIs	70
6.1	APIs by Category	70
6.1.1	Library and Plugin APIs	70
6.1.2	Node APIs	71
6.1.3	Logical HBA APIs	71
6.1.4	Physical HBA APIs	72
6.1.5	Network Portal APIs	73
6.1.6	Logical Network Port (LNP) APIs	73
6.1.7	Physical Network Port (PNP) APIs	73
6.1.8	Target APIs	73
6.1.9	Logical Unit (LU) APIs	74
6.1.10	Miscellaneous APIs	75
6.2	APIs by Name	76
6.2.1	IMA_AddDiscoveryAddress.....	78
6.2.2	IMA_AddStaticDiscoveryTarget	79
6.2.3	IMA_DeregisterForObjectPropertyChanges	81
6.2.4	IMA_DeregisterForObjectVisibilityChanges.....	81
6.2.5	IMA_ExposeLu	82
6.2.6	IMA_FreeMemory	83
6.2.7	IMA_GenerateNodeName	84
6.2.8	IMA_GetAddressKeys.....	85
6.2.9	IMA_GetAssociatedPluginOid	86
6.2.10	IMA_GetDataPduInOrderProperties	87
6.2.11	IMA_GetDataSequenceInOrderProperties.....	88
6.2.12	IMA_GetDefaultTime2RetainProperties	89
6.2.13	IMA_GetDefaultTime2WaitProperties.....	90
6.2.14	IMA_GetDeviceStatistics	91
6.2.15	IMA_GetDiscoveryAddressOidList	92
6.2.16	IMA_GetDiscoveryAddressProperties	93
6.2.17	IMA_GetDiscoveryProperties.....	94

6.2.18	IMA_GetErrorRecoveryLevelProperties	95
6.2.19	IMA_GetFirstBurstLengthProperties	96
6.2.20	IMA_GetImmediateDataProperties	97
6.2.21	IMA_GetInitialR2TProperties	98
6.2.22	IMA_GetInitiatorAuthParms	99
6.2.23	IMA_GetInUseInitiatorAuthMethods	100
6.2.24	IMA_GetIpProperties	101
6.2.25	IMA_GetIpsecProperties	102
6.2.26	IMA_GetLhbaOidList	103
6.2.27	IMA_GetLhbaProperties	104
6.2.28	IMA_GetLibraryProperties	105
6.2.29	IMA_GetLnpOidList	105
6.2.30	IMA_GetLnpProperties	106
6.2.31	IMA_GetLuOid	107
6.2.32	IMA_GetLuOidList	108
6.2.33	IMA_GetLuProperties	109
6.2.34	IMA_GetMaxBurstLengthProperties	110
6.2.35	IMA_GetMaxConnectionsProperties	111
6.2.36	IMA_GetMaxOutstandingR2TProperties	112
6.2.37	IMA_GetMaxRecvDataSegmentLengthProperties	113
6.2.38	IMA_GetNetworkPortalOidList	114
6.2.39	IMA_GetNetworkPortalProperties	115
6.2.40	IMA_GetNetworkPortStatus	115
6.2.41	IMA_GetNodeProperties	116
6.2.42	IMA_GetNonSharedNodeOidList	117
6.2.43	IMA_GetObjectType	118
6.2.44	IMA_GetPhbaDownloadProperties	119
6.2.45	IMA_GetPhbaOidList	120
6.2.46	IMA_GetPhbaProperties	121
6.2.47	IMA_GetPhbaStatus	122
6.2.48	IMA_GetPluginOidList	122
6.2.49	IMA_GetPluginProperties	123
6.2.50	IMA_GetPnpOidList	124
6.2.51	IMA_GetPnpProperties	125
6.2.52	IMA_GetPnpStatistics	126
6.2.53	IMA_GetSharedNodeOid	127
6.2.54	IMA_GetStaticDiscoveryTargetOidList	128
6.2.55	IMA_GetStaticDiscoveryTargetProperties	129
6.2.56	IMA_GetStatisticsProperties	130
6.2.57	IMA_GetSupportedAuthMethods	131
6.2.58	IMA_GetTargetErrorStatistics	132
6.2.59	IMA_GetTargetOidList	133
6.2.60	IMA_GetTargetProperties	135
6.2.61	IMA_IsPhbaDownloadFile	136
6.2.62	IMA_LuInquiry	137
6.2.63	IMA_LuReadCapacity	139
6.2.64	IMA_LuReportLuns	141
6.2.65	IMA_PhbaDownload	143
6.2.66	IMA_PluginIOCtl	144

6.2.67	IMA_RegisterForObjectPropertyChanges	146
6.2.68	IMA_RegisterForObjectVisibilityChanges	147
6.2.69	IMA_RemoveDiscoveryAddress	148
6.2.70	IMA_RemoveStaleData	149
6.2.71	IMA_RemoveStaticDiscoveryTarget	150
6.2.72	IMA_SetDataPduInOrder	151
6.2.73	IMA_SetDataSequenceInOrder	152
6.2.74	IMA_SetDefaultGateway	153
6.2.75	IMA_SetDefaultTime2Retain	154
6.2.76	IMA_SetDefaultTime2Wait	155
6.2.77	IMA_SetDnsServerAddress	156
6.2.78	IMA_SetErrorRecoveryLevel	157
6.2.79	IMA_SetFirstBurstLength	159
6.2.80	IMA_SetImmediateData	160
6.2.81	IMA_SetInitialR2T	161
6.2.82	IMA_SetInitiatorAuthMethods	162
6.2.83	IMA_SetInitiatorAuthParms	163
6.2.84	IMA_SetIpConfigMethod	164
6.2.85	IMA_SetIsnsDiscovery	166
6.2.86	IMA_SetMaxBurstLength	167
6.2.87	IMA_SetMaxConnections	169
6.2.88	IMA_SetMaxRecvDataSegmentLength	170
6.2.89	IMA_SetMaxOutstandingR2T	171
6.2.90	IMA_SetNetworkPortallpAddress	172
6.2.91	IMA_SetNodeAlias	173
6.2.92	IMA_SetNodeName	174
6.2.93	IMA_SetSendTargetsDiscovery	176
6.2.94	IMA_SetSlpDiscovery	177
6.2.95	IMA_SetStaticDiscovery	178
6.2.96	IMA_SetStatisticsCollection	179
6.2.97	IMA_SetSubnetMask	181
6.2.98	IMA_UnexposeLu	182
7	Implementation compliance	183
8	Notes	183
8.1	Client Usage Notes	183
8.1.1	Persisted Object IDs	183
8.1.2	Reserved Fields	183
8.1.3	Event Notification Within a Single Client	183
8.1.4	Event Notification and Multi-Threading	184
8.1.5	IPsec Security	184
8.1.6	Transmission of Authorization Parameters	184
8.1.7	Target OIDs and iSCSI Targets	184
8.1.8	Configuration Changes and the IMA_STATUS_REBOOT_NECESSARY status	184
8.2	Library Implementation Notes	184
8.2.1	Object IDs	184
8.2.2	Multi-threading Support	184
8.2.3	Event Notification and Multi-Threading	184
8.2.4	Structure Packing	184

8.2.5	Calling Conventions.....	185
8.3	Plugin Implementation Notes	185
8.3.1	Object IDs	185
8.3.2	Reserved Fields	185
8.3.3	Multi-threading Support	185
8.3.4	Event Notification to Different Clients	185
8.3.5	Event Notification and Multi-Threading	185
8.3.6	IPsec Security	185
8.3.7	Persistence of Authorization Parameters	186
8.3.8	Executing SCSI Commands and Operating System Compatibility	186
8.3.9	Executing SCSI Commands and Session Management.....	186
8.3.10	Plugin IOCTLs	186
8.3.11	Target OIDs and Logical Unit OIDs.....	186
Annex A (informative)	Device Names.....	187
Annex B (informative)	Coding Examples	188
Bibliography		198

INTERNATIONAL ELECTROTECHNICAL COMMISSION

INFORMATION TECHNOLOGY –

iSCSI management API

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 11989 was prepared by subcommittee 25: Interconnection of information technology equipment, of ISO/IEC joint technical committee 1: Information technology.

International Standard ISO/IEC 11989 was prepared by ANSI, was adopted, under the fast track procedure, by joint technical committee 1: Information technology and has been assigned to subcommittee 25: Interconnection of information technology equipment.

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.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

INFORMATION TECHNOLOGY –

iSCSI management API

1 Scope

This International Standard specifies an Application Programming Interface (API) that provides interfaces to discover and manage iSCSI resources on a system. This International Standard is applicable to vendors who deliver drivers that provide iSCSI resources to a system.

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/IEC. In particular, it does not give the referenced specification the status of an International Standard.

ISO/IEC 14776-453, *Information technology – Small Computer System Interface – Part 453: SCSI Primary Commands-3 (SPC-3)*

ISO/IEC 19501, *Unified Modeling Language (UML)*

ISO/IEC 8802-3:2000, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*¹

IETF RFC 1994, *PPP Challenge Handshake Authentication Protocol (CHAP)*

IETF RFC 2945, *The SRP Authentication and Key Exchange System*

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

ANSI INCITS 386-2004, *Fibre Channel HBA API (FC-HBA)*

3 Document conventions

Each API's description is divided into seven sections. These sections are described below.

1. Synopsis

This section gives a brief description of what action the API performs.

2. Prototype

¹ IEEE 802.1AX, IEEE Standard for Information technology

This section gives a C prototype of the function. The prototypes show the following.

- The name of the API.
- The return type of the API.

3. Parameters

This section lists each parameter along with an explanation of what the parameter represents.

4. Typical Return Values

This section lists the Typical Return Values of the API with an explanation of why a particular return value would be returned. It is important to note that this list is **not** a comprehensive list of all of the possible return values. There are certain errors, e.g. [IMA_ERROR_INSUFFICIENT_MEMORY](#), that might be returned by any API. Return values such as these are not listed.

5. Remarks

This section contains comments about the API that may be useful to the reader. In particular, this section will contain extra information about the information returned by the API.

6. Support

This section states whether an API is mandatory to be supported, optional to be supported, or mandatory to be supported under certain conditions.

- If an API is mandatory to be supported a client can rely on the API functioning under all circumstances.
- If the API is optional to be supported then a client cannot rely on the API functioning.
- If the API is mandatory to be supported under certain conditions then a client can rely on the API functioning if the specified conditions are met. Otherwise a client should assume that the API is not supported.

7. See Also

This section lists other related APIs or related code examples that the reader might find useful.

4 Background technical information

4.1 Terms, definitions and abbreviations

For the purposes of this document the following terms, definitions and abbreviations apply.

4.1.1

Alias

an alias string can be associated with an iSCSI Node. The alias allows an organization to associate a user-friendly string with the iSCSI Name. However, the alias string is not a substitute for the iSCSI Name

NOTE Definition taken from IETF RFC 3720.

4.1.2

Discovery Address

an address used in a SendTargets discovery session. The discovery address is used to represent one or more targets to be discovered. One example of a discovery address is a gateway that exposes one or more targets to one or more iSCSI initiators

4.1.3

Host

a compute node connected to the SAN

4.1.4

iSCSI Node

the iSCSI Node represents a single iSCSI initiator or iSCSI target. There are one or more iSCSI Nodes within a Network Entity. The iSCSI Node is accessible via one or more Network Portals. An iSCSI Node is identified by its iSCSI Name. The separation of the iSCSI Name from the addresses used by and for the iSCSI node allows multiple iSCSI nodes to use the same address, and the same iSCSI node to use multiple addresses

NOTE Definition taken from IETF RFC 3720.

4.1.5

Logical HBA

a representation of a parallel SCSI HBA to the operating system

4.1.6

Logical Network Port

a logical network port is a collection of one or more physical network ports that have been aggregated together. This can be done using link aggregation (see IEEE Std 802.1AX, but may be done in other ways as well

NOTE If more than one physical network port is used to create a logical network port those physical network ports do not have to be on the same PHBA.

4.1.7

Network Entity

the Network Entity represents a device or gateway that is accessible from the IP network. A Network Entity must have one or more Network Portals, each of which can be used to gain access to the IP network by some iSCSI Nodes contained in that Network Entity

NOTE Definition taken from IETF RFC 3720.

4.1.8

Network Portal

the Network Portal is a component of a Network Entity that has a TCP/IP network address and that may be used by an iSCSI Node within that Network Entity for the connection(s) within one

of its iSCSI sessions. A Network Portal in an initiator is identified by its IP address. A Network Portal in a target is identified by its IP address and its listening TCP port

NOTE Definition taken from IETF RFC 3720.

4.1.9

Object ID

a unique identifier assigned to any object within the IMA. Objects sometimes represent physical entities, e.g. physical HBAs. At other times, objects represent logical entities, e.g. network portals

4.1.10

Persistent

the quality of something being non-volatile. This usually means that it is recorded on some non-volatile medium such as flash RAM or magnetic disk. Implicitly, this is also readable from the non-volatile medium

Examples of persistent storage:

- under Windows, the Registry would be a common place to find persistently stored values (assuming that the values are not stored as volatile);
- under any OS a file on magnetic hard disk would be persistent

4.1.11

Physical HBA

a physical HBA (PHBA) is a controller card that has one or more physical network ports mounted on it and that plugs into a slot in a motherboard. If a motherboard has physical network ports mounted on it directly, it can be considered a PHBA *in regards to the requirements specified in this document*. Normally, a motherboard would not be considered a PHBA

4.1.12

Physical Network Port

a physical connection on a physical HBA that connects the PHBA to the network, e.g. an RJ-45. A physical HBA has one or more physical network ports

4.1.13

Plugin

a plugin is software, typically written by an HBA vendor, that provides support for one or more models of iSCSI HBAs. The plugin's job is to provide a bridge between the library's interface and the vendor's HBA device driver. A plugin is implemented as a loadable module: a DLL in Windows and a shared object in UNIX. A plugin is accessed by an application through the iSCSI Management API library.

The SNIA FC HBA API's concept of a vendor library is the equivalent to a plugin

4.1.14

Portal Groups

iSCSI supports multiple connections within the same session; some implementations will have the ability to combine connections in a session across multiple Network Portals. A Portal Group defines a set of Network Portals within an iSCSI Network Entity that collectively supports the capability of coordinating a session with connections spanning these portals. Not all Network Portals within a Portal Group need participate in every session connected through that Portal Group. One or more Portal Groups may provide access to an iSCSI Node. Each Network Portal, as utilized by a given iSCSI Node, belongs to exactly one portal group within that node

NOTE Definition taken from IETF RFC 3720.

4.1.15

Portal Group Tag

this 16-bit quantity identifies the Portal Group within an iSCSI Node. All Network Portals with the same portal group tag in the context of a given iSCSI Node are in the same Portal Group

NOTE Definition taken from IETF RFC 3720.

4.1.16

Primary Discovery Method

a discovery method that does not depend upon any other discovery method to discover targets. iSNS target discovery, SLP target discovery, and static target discovery are all primary discovery methods

4.1.17

Secondary Discovery Method

a discovery method that depends upon other discovery methods to discover targets. SendTargets target discovery is a secondary discovery method because it cannot discover any targets without using some other discovery method

4.1.18

Stale Data

stale data is configuration data that refers to targets or logical units that are no longer present or that are no longer visible to the system.

For example, setting iSCSI login parameters associated with a target using the [IMA_SetFirstBurstLength](#) API and then removing the target from the network will create stale data because the configuration data that's been set for the target is retained after the target is removed from the network. Similarly, calling the [IMA_ExposeLu](#) API for a logical unit and then removing the logical unit's target from the network will create stale data because configuration data indicating that the logical unit is exposed is retained after the target is removed from the network

4.1.19

Unicode

unicode is a system of uniquely identifying (numbering) characters such that nearly any character in any language is identified

4.1.20

UTF-8

unicode Transformation Format, 8-bit encoding form. UTF-8 is the Unicode Transformation Format that serializes a Unicode scalar value as a sequence of one to four bytes

NOTE Definition taken from the glossary of the Unicode Consortium web site. See <http://www.unicode.org/glossary/index.html>.

4.1.21 Abbreviations

API	A pplication P rogramming I nterface
FC	F ibre C hannel
ID	I Dentifier
iSCSI	internet S mall C omputer S ystem I nterface
iSNS	internet S torage N ame S ervice
HBA	H ost B us A daptor
IMA	iSCSI M anagement A PI

LHBA	Logical Host Bus Adaptor
LNP	Logical Network Port
LUN	Logical Unit Number
RAM	Random-Access Memory
OID	Object ID
PHBA	Physical Host Bus Adaptor
PNP	Physical Network Port
SLP	Service Location Protocol
SNIA	Storage Networking Industry Association
UTF	Unicode Transformation Format

4.2 Concepts

4.2.1 Library and plugins

The iSCSI Management API shall be implemented in one of two ways:

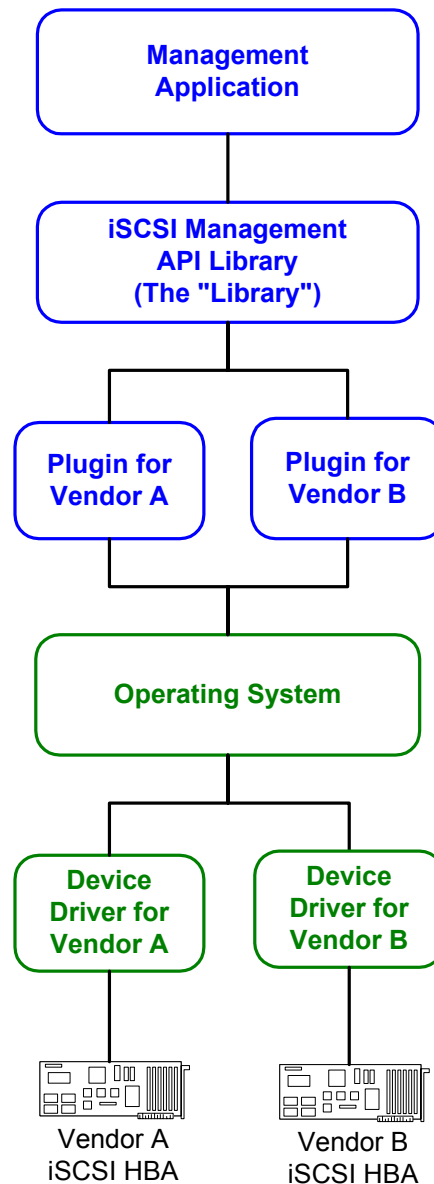
- a library;
- a library in combination with plugins.

If an implementation uses a library without plugins then either the plugin functionality is built into the library itself or the library is able to interface with vendor specific modules using a pre-existing interface. These vendor specific modules would provide functionality equivalent to plugins.

The library provides an interface that applications use to perform iSCSI management. Among other things, the library is responsible for loading plugins and dispatching requests from a management application to the appropriate plugin(s).

Plugins are provided by iSCSI HBA vendors to manage their hardware. Typically, a plugin will take a request in the generic format provided by the library and then translate that request into a vendor specific format and forward the request onto the vendor's device driver. In practice, a plugin may use a DLL or shared object library to communicate with the device driver. Also, it may communicate with multiple device drivers. Ultimately, the method a plugin uses to accomplish its work is entirely vendor specific. It is anticipated that most implementations will use plugins and such an implementation is generally assumed throughout this document. With the exception of two APIs ([IMA_GetAssociatedPluginOid](#) and [IMA_GetPluginOidList](#)), the method of implementation does not matter to the client.

The figure below shows a simple block diagram of how the iSCSI Management API library and plugins fit into a total iSCSI management application architecture.



4.2.2 Object ID

The core element of the iSCSI Management API (IMA) is the object ID (OID). An object ID is a structure that “uniquely” identifies an object. The reason uniquely is in quotes in the previous sentence is that it is possible, though very unlikely, that an object ID would be reused and refer to a different object.

An object ID consists of three fields:

- 1) An object type. This identifies the type of object, e.g. iSCSI node, PHBA, LHBA, etc., that the object ID refers to.
- 2) An object owner identifier. This is a number that is used to uniquely identify the owner of the object. Objects are owned by either the library or a plugin.
- 3) An object sequence number. This is a number used by the owner of an object, possibly in combination with the object type, to identify an object.

To a client that uses the library object IDs shall be considered opaque. A client shall use only documented APIs to access information found in the object ID.

There are several rules for object IDs that the library, plugins, and clients shall follow. They are:

An object ID can only refer to one object at a time.

An object can only have one object ID that refers to it at any one time. It is not permissible to have two or more object IDs that refer to the same object at the same time. In some cases this may be difficult, but the rule still shall be followed.

For example, suppose a PHBA is in a system. That PHBA will have an object ID. If the PHBA is removed and then reinserted (while the associated plugin is running) then one of two things can happen:

- The PHBA can retain the same object ID as it had before it was removed
- OR
- The PHBA can get a new object ID and the old object ID will no longer be usable.

This can only happen if the *same* PHBA is reinserted. If a PHBA is removed and another PHBA is inserted that has not been in the system while a particular instance of the library and plugins are running then that PHBA *shall* be given a new object ID.

The library and plugins can uniquely identify an object within their own object space by using either the object sequence number or by using the object sequence number in combination with the object type. Which method is used is up to the implementer of the library or plugin.

Object sequence numbers shall be reused in a conservative fashion to minimize the possibility that an object ID will ever refer to two (or more) different objects in any once instance of the library or plugin. This rule for reuse only applies to a particular instance of the library or plugin. Neither the library nor plugins are required or expected to persist object sequence numbers across instances.

Because neither the library nor plugins are required to persist object sequence numbers a client using the library shall not use persisted object IDs across instances of itself.

Similarly, different instances of the library and plugins *may* use different object IDs to represent the same physical entity.

4.2.3 Object ID list

An object ID list is a list of zero or more object IDs. There are several APIs, e.g. [IMA_GetNonSharedNodeOidList](#), that return object ID lists. Once a client is finished using an object ID list the client shall free the memory used by the list by calling the [IMA_FreeMemory](#) API.

4.2.4 The Shared Node versus Non-shared Nodes

The following is the definition of an iSCSI node found in IETF RFC 3720:

The iSCSI Node represents a single iSCSI initiator or iSCSI target. There are one or more iSCSI Nodes within a Network Entity. The iSCSI Node is accessible via one or more Network Portals. An iSCSI Node is identified by its iSCSI Name. The separation of the iSCSI Name from the addresses used by and for the iSCSI node allows multiple iSCSI nodes to use the same addresses, and the same iSCSI node to use multiple addresses. iSCSI nodes also have addresses.

The following text from IETF RFC 3720 gives a somewhat clearer idea of what an iSCSI node is:

An iSCSI name really names a logical software entity, and is not tied to a port or other hardware that can be changed. For instance, an initiator name should name the iSCSI initiator node, not a particular NIC or HBA. When multiple NICs are used, they should generally all present the same iSCSI initiator name to the targets, because they are just paths to the same SCSI layer. In most operating systems, the named entity is the operating system image.

So, in simple terms, an iSCSI node is a uniquely named entity that runs on a particular operating system image. IETF RFC 3720 allows for more than one node to run on a particular image, but it strongly encourages that only one node be used. (For more information on why this is the case please refer to the various iSCSI related specifications.)

To this end the iSCSI Management API has the concept of a shared node. The shared node is intended to be the single node of an operating system image that is encouraged in the iSCSI specification. The reason that it's called the shared node is that the node is shared by all the vendors whose iSCSI HBAs are in a system.

All other nodes in an operating system image are considered to be non-shared nodes. That is, they are nodes that are created by an HBA vendor to be used exclusively by that vendor's iSCSI HBAs in combination with the operating system image. Non-shared nodes run counter to the spirit of what IETF RFC 3720 intended a node to be. However, the need for them in this specification reflects some limitations of existing HBA implementations.

4.2.5 Logical HBA

A logical HBA (LHBA) is a representation of a parallel SCSI HBA to the operating system. Typically, today's operating systems only have an understanding of the specifics of parallel SCSI. They don't understand other SCSI transports such as FCP or iSCSI. Therefore, device drivers for these other transports are required to map transport specific concepts to parallel SCSI concepts. For example, an iSCSI HBA device driver cannot identify a device to the OS using the iSCSI node name. Instead, it must conjure up a parallel SCSI ID (0-15) for the device. In addition, it may have to map eight byte SCSI LUNs to three *bit* SCSI LUNs.

In addition, some HBA vendors provide dynamic multi-pathing in their device drivers. This allows the host to communicate with targets using different initiator ports in the host, thus allowing the device driver to provide both load balancing and fail over capabilities transparently to the operating system. An LHBA allows a device driver to easily implement these features transparently to the operating system.

4.2.6 Target OIDs and Logical Unit OIDs

A target OID should not be confused with a parallel SCSI target ID. A target OID is an [IMA_OID](#) structure in which the object type field indicates that the OID structure specifies an iSCSI target accessible via a LHBA. An iSCSI target will have a unique OID on each LHBA that can access a LU of the target.

Similarly, a logical unit OID should not be confused with a SCSI LUN. A logical unit OID is an [IMA_OID](#) structure in which the object type field indicates that the OID structure specifies an iSCSI logical unit accessible via an iSCSI target. A logical unit will have a unique OID on each LHBA that can access the LU.

4.2.7 Software Initiators versus Hardware Initiators

There are two basic types of implementations of iSCSI initiators, they are typically referred to as software initiators and hardware initiators.

- A software initiator usually is a device driver that runs on top of the TCP/IP stack that comes with the operating system. It connects to iSCSI targets using the NICs that are being used for traditional networking tasks.
- A hardware initiator usually includes a specialized adapter, usually referred to as an HBA, that includes special hardware and/or firmware for accelerating TCP/IP and sometimes iSCSI as well. This solution also includes a device driver to allow the operating system to use the HBA.

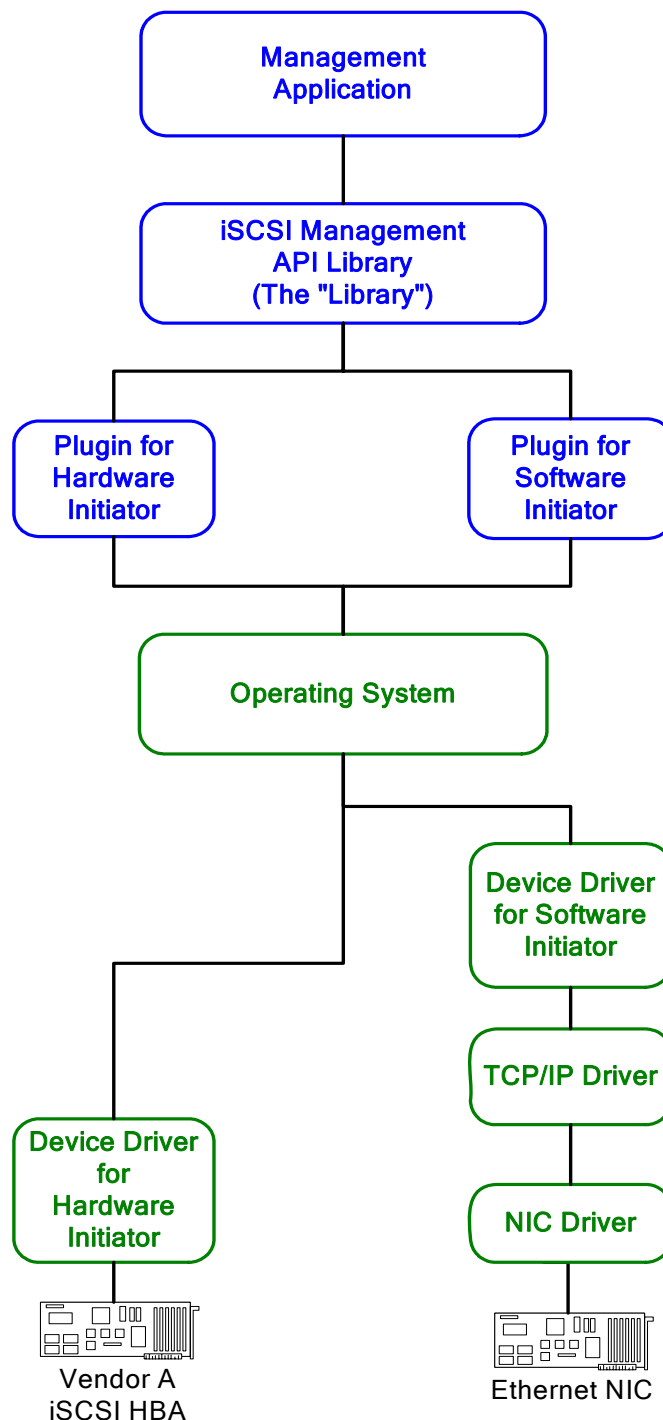
These terms are imprecise as both types of initiators use software and hardware. However, these terms have come into common use and so this specification uses them as well.

The reason to introduce these two types of initiators is that they provide different levels of discovery and management capabilities, which requires clients to manage them somewhat differently.

For example, a software initiator will typically know little, if anything, about the underlying hardware that is being used. Therefore, an IMA client cannot determine (using IMA) the topology of the storage network, the redundancy of connections between the initiator and a target, etc. Meanwhile, a hardware initiator will know the hardware that it is using and be able to provide an IMA client with this kind of information.

Another example, is that when querying and setting various discovery methods for iSCSI targets a software initiator will require this be done using the logical HBA, while a hardware initiator may allow either the logical or the physical HBA to be used. The software initiator requires that the logical HBA be used because it doesn't know anything about the underlying hardware and it cannot therefore expose any physical HBA objects.

The diagram below shows the software and hardware stack needed for both a hardware initiator and a software initiator.



4.2.8 iSCSI session and connection parameters

iSCSI supports the negotiation of both session specific and connection specific parameters, e.g., max burst length. This API recognizes two levels at which a client can query and set these parameters: the target level, the LHBA level. There is a third level which is used, but is not queryable or settable: the driver level.

If a client sets one or more of these parameters at the target level then those values will be used as the proposed initial value when negotiating the actual value to use at runtime with a

target. A client can only set a proposed initial value for a parameter; it cannot specify an actual value that is guaranteed to be used.

If a client sets a value for a parameter at the LHBA level, then that value will be used as the proposed initial value when negotiating the actual value to use at runtime *unless* a value for that parameter has been set for that parameter on that specific target. Thus a target specific value overrides the setting of an LHBA specific value.

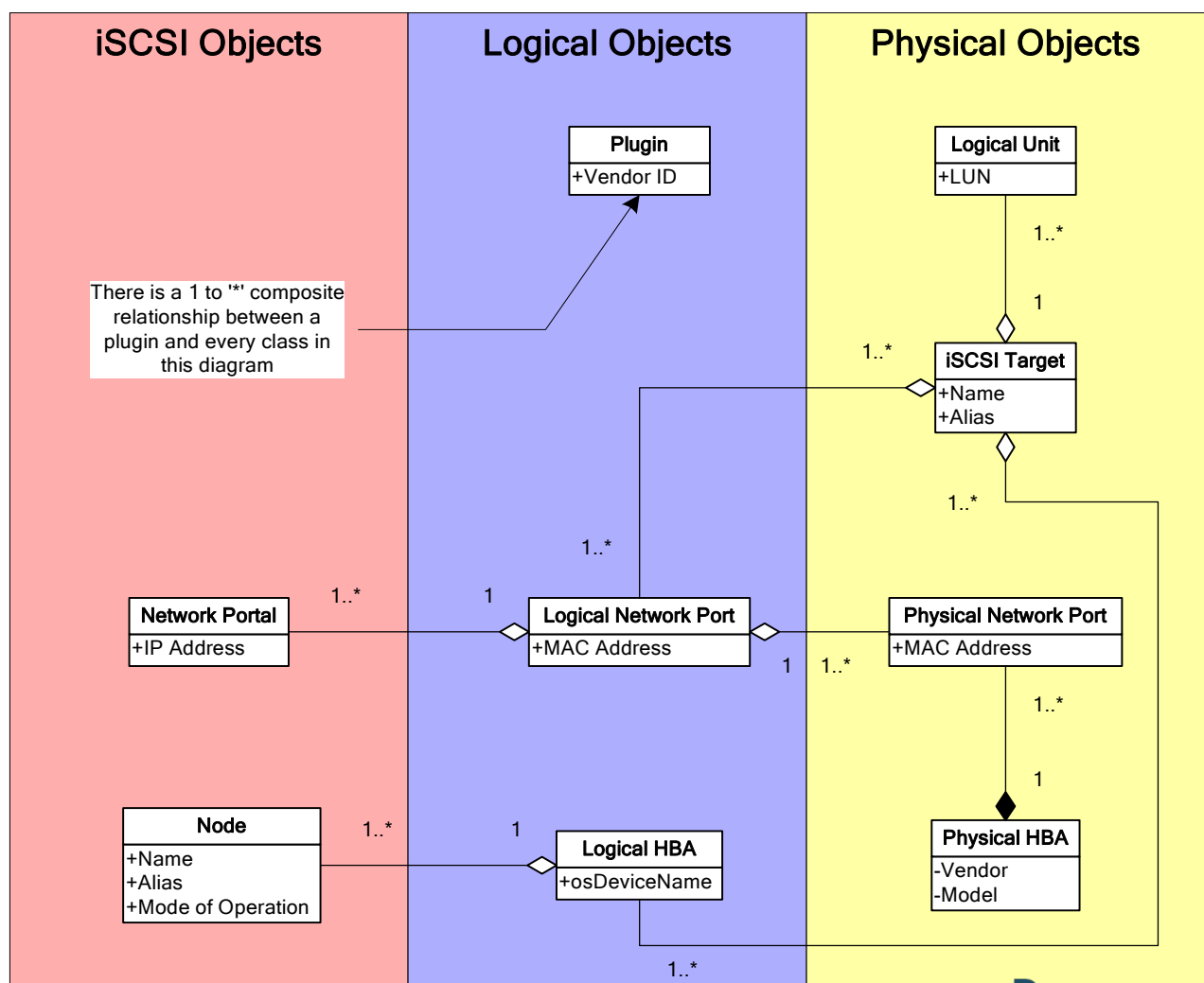
If a client does not set a value for a parameter at either the target or the LHBA level then the driver level value, i.e., a default value specified by the driver implementation is used.

In summary:

- If a target level value for a parameter has been specified it is used.
- If no target level value has been specified for a parameter, but an LHBA level value has been specified then it is used.
- Finally, if neither a target level nor an LHBA level value for a parameter has been specified then the driver level (implementation default) value is used.

4.2.9 Class relationship diagram

Below is a Universal Modeling Language (ISO/IEC 19501 UML) diagram that shows the relationship between the various classes of objects in the IMA. Each class may contain a few example properties.



5 Constants and types

5.1 IMA_WCHAR

Typedef'd as a `wchar_t`.

5.2 IMA_BYTE

The smallest unsigned integer that is at least 8 bits in length.

5.3 IMA_BOOL

Typedef'd to an `IMA_UINT32`. A variable of this type can have either of the following values:

- `IMA_TRUE`
This symbol has the value 1.
- `IMA_FALSE`
This symbol has the value 0.

5.4 IMA_XBOOL

Typedef'd to an `IMA_UINT32`. This is an extended boolean. A variable of this type can have any of the following values:

- `IMA_TRUE`
This symbol has the value 1.
- `IMA_FALSE`
This symbol has the value 0.
- `IMA_UNKNOWN`
This symbol has the value 0xFFFFFFFF.

5.5 IMA_UINT

The smallest unsigned integer that is at least 32 bits in length.

5.6 IMA_UINT16

The smallest unsigned integer that is at least 16 bits in length.

5.7 IMA_UINT32

The smallest unsigned integer that is at least 32 bits in length.

5.8 IMA_UINT64

The smallest unsigned integer that is at least 64 bits in length.

5.9 IMA_DATETIME

Typedef'd to a `struct tm`. This is a structure declared in `time.h` that comes with the standard C runtime library.

5.10 IMA_OBJECT_VISIBILITY_FN

5.10.1.1 Format

```
typedef void (* IMA_OBJECT_VISIBILITY_FN)(
    /* in */ IMA_BOOL becomingVisible,
    /* in */ IMA_OID oid
);
```

5.10.1.2 Parameters

becomingVisible

A boolean indicating if the object specified by *oid* is becoming visible or is disappearing. If this parameter has the value IMA_TRUE then the object is becoming visible. If this parameter has the value IMA_FALSE then the object is disappearing.

oid

The object ID of the object whose visibility is changing.

5.10.1.3 Remarks

This type is used to declare client functions that can be used with the [IMA_RegisterForObjectVisibilityChanges](#) and [IMA_DeregisterForObjectVisibilityChanges](#) APIs.

5.11 IMA_OBJECT_PROPERTY_FN

5.11.1.1 Format

```
typedef void (* IMA_OBJECT_PROPERTY_FN)(
    /* in */ IMA_OID oid
);
```

5.11.1.2 Parameters

oid

The object ID of the object whose property(ies) changed.

5.11.1.3 Remarks

This type is used to declare client functions that can be used with the [IMA_RegisterForObjectPropertyChanges](#) and [IMA_DeregisterForObjectPropertyChanges](#) APIs.

5.12 IMA_OBJECT_TYPE

5.12.1 Format

```
typedef enum IMA_object_type
{
    IMA_OBJECT_TYPE_UNKNOWN           = 0,
    IMA_OBJECT_TYPE_PLUGIN            = 1,
    IMA_OBJECT_TYPE_NODE              = 2,
    IMA_OBJECT_TYPE_LHBA              = 3,
    IMA_OBJECT_TYPE_PHBA              = 4,
    IMA_OBJECT_TYPE_NETWORK_PORTAL    = 5,
    IMA_OBJECT_TYPE_PORTAL_GROUP      = 6,
    IMA_OBJECT_TYPE_LNP               = 7,
    IMA_OBJECT_TYPE_PNP               = 8,
    IMA_OBJECT_TYPE_TARGET            = 9,
    IMA_OBJECT_TYPE_LU                = 10,
    IMA_OBJECT_TYPE_DISCOVERY_ADDRESS = 11,
    IMA_OBJECT_TYPE_STATIC_DISCOVERY_TARGET = 12
} IMA_OBJECT_TYPE;
```

5.12.2 Fields

5.12.2.1 IMA_OBJECT_TYPE_UNKNOWN

The object has an unknown type. If an object has this type it's most likely an uninitialized object.

This symbol has the value 0.

5.12.2.2 IMA_OBJECT_TYPE_PLUGIN

The object represents a plugin to the IMA library.

This symbol has the value 1.

5.12.2.3 IMA_OBJECT_TYPE_NODE

The object represents an iSCSI node.

This symbol has the value 2.

5.12.2.4 IMA_OBJECT_TYPE_LHBA

The object represents a logical HBA.

This symbol has the value 3.

5.12.2.5 IMA_OBJECT_TYPE_PHBA

The object represents a physical HBA.

This symbol has the value 4.

5.12.2.6 IMA_OBJECT_TYPE_NETWORK_PORTAL

The object represents an iSCSI network portal.

This symbol has the value 5.

5.12.2.7 IMA_OBJECT_TYPE_PORTAL_GROUP

The object represents an iSCSI portal group.

This symbol has the value 6.

5.12.2.8 IMA_OBJECT_TYPE_LNP

The object represents a logical network port.

This symbol has the value 7.

5.12.2.9 IMA_OBJECT_TYPE_PNP

The object represents a physical network port.

This symbol has the value 8.

5.12.2.10 IMA_OBJECT_TYPE_TARGET

The object represents an iSCSI target relative to an LHBA.

This symbol has the value 9.

5.12.2.11 IMA_OBJECT_TYPE_LU

The object represents a logical unit relative to a target.

This symbol has the value 10.

5.12.2.12 IMA_OBJECT_TYPE_DISCOVERY_ADDRESS

The object represents a discovery address relative to a PNP or an LHBA.

This symbol has the value 11.

5.12.2.13 IMA_OBJECT_TYPE_STATIC_DISCOVERY_TARGET

The object represents a static discovery target relative to a LNP or an LHBA.

This symbol has the value 12.

5.13 IMA_STATUS

IMA_STATUS is an enumerated type used to indicate the status of an API call. Most statuses are errors, however some are not. The non-error statuses indicate that an operation successfully completed. However, there is additional information that the client needs to be aware of and the status indicates what this information is.

Currently there are a limited number of status values that indicate that an invalid parameter was specified to an API call. Additional statuses which indicate more precisely why a parameter was invalid may be added to future versions of the iSCSI Management API specification. Statuses in the range of 0xC0000000 to 0xCFFFFFFF are reserved for indicating conditions that mean an invalid parameter was specified to an API. Clients shall be written to handle invalid parameter statuses that may be added in later versions of this specification.

5.13.1 Macros

5.13.1.1 IMA_SUCCESS

This macro returns IMA_TRUE if the specified status code indicates that a call succeeded. It returns IMA_FALSE if the specified status code indicates that the call failed.

5.13.1.2 IMA_ERROR

This macro returns IMA_TRUE if the specified status code indicates that a call failed. It returns IMA_FALSE if the specified status code indicates that the call succeeded.

5.13.2 Non-error statuses

5.13.2.1 IMA_STATUS_SUCCESS

This status indicates that the API call succeeded.

This symbol has the value 0x00000000.

5.13.2.2 IMA_STATUS_REBOOT_NECESSARY

This status indicates that the operation succeeded, but a reboot is necessary to have the change take affect.

This symbol has the value 0x00000001.

5.13.2.3 IMA_STATUS_INCONSISTENT_NODE_PROPERTIES

This status indicates there is an inconsistency between the node properties, specifically either the node name or node alias, kept by the library and one or more plugins. The client should set both the node name (using [IMA_SetNodeName](#)) and node alias (using [IMA_SetNodeAlias](#)) to fix this problem.

This symbol has the value 0x00000002.

5.13.2.4 IMA_STATUS_SCSI_STATUS_CONDITION_MET

This status indicates that a SCSI command succeeded with a CONDITION MET status.

This symbol has the value 0x00000100.

5.13.3 Error statuses

5.13.3.1 IMA_ERROR_NOT_SUPPORTED

This error indicates that the specified API is not supported by the owner of the object.

This symbol has the value 0x80000001.

5.13.3.2 IMA_ERROR_INSUFFICIENT_MEMORY

This error indicates that there was insufficient memory to complete the request. It is possible that any API can return this error.

This symbol has the value 0x80000002.

5.13.3.3 IMA_ERROR_LAST_PRIMARY_DISCOVERY_METHOD

This error indicates that the call would disable the last primary discovery method for the PHBA specified in the call. A client is not allowed to disable all primary discovery methods for a PHBA.

This symbol has the value 0x80000003.

5.13.3.4 IMA_ERROR_UNEXPECTED_OS_ERROR

This error indicates that either the library or plugin encountered an unexpected error from an OS API while attempting to perform the requested operation.

This symbol has the value 0x80000004.

5.13.3.5 IMA_ERROR_SYNC_TIMEOUT

This error indicates that an attempt to acquire ownership of some synchronization mechanism, e.g. a mutex or semaphore, has timed out.

This symbol has the value 0x80000005.

5.13.3.6 IMA_ERROR_LU_EXPOSED

This error indicates the requested operation cannot be completed because an LU is currently exposed to the operating system. For the called API to succeed the logical unit shall not be exposed to the operating system. This error is returned by the [IMA_ExposeLu](#) and [IMA_RemoveStaticDiscoveryTarget](#) APIs.

This symbol has the value 0x80000006.

5.13.3.7 IMA_ERROR_LU_NOT_EXPOSED

This error indicates an attempt to use a logical unit that is not currently exposed to the operating system. For the called API to succeed the logical unit shall be exposed to the operating system. This error is returned by the [IMA_UnexposeLu](#) and the [IMA_GetDeviceStatistics](#) APIs.

This symbol has the value 0x80000007.

5.13.3.8 IMA_ERROR_LU_IN_USE

This error indicates an attempt to unexpose a logical unit that is in use by the operating system. This error is returned by the [IMA_UnexposeLu](#) API.

This symbol has the value 0x80000008.

5.13.3.9 IMA_ERROR_TARGET_TIMEOUT

This error indicates that communication with a target was necessary to perform the requested API and that the target didn't respond to a command that was sent to it.

This symbol has the value 0x80000009.

5.13.3.10 IMA_ERROR_LOGIN_REJECTED

This error indicates that a login to a target was needed to perform the requested API and the target rejected the attempt.

This symbol has the value 0x8000000A.

5.13.3.11 IMA_ERROR_STATS_COLLECTION_NOT_ENABLED

This error indicates that an attempt was made to retrieve statistics from an object that did not have statistics collection enabled.

This symbol has the value 0x8000000B.

5.13.3.12 IMA_ERROR_SCSI_STATUS_CHECK_CONDITION

This error indicates that a SCSI command failed with a CHECK CONDITION status.

This symbol has the value 0x80000100.

5.13.3.13 IMA_ERROR_SCSI_STATUS_BUSY

This error indicates that a SCSI command failed with a BUSY status.

This symbol has the value 0x80000101.

5.13.3.14 IMA_ERROR_SCSI_STATUS_RESERVATION_CONFLICT

This error indicates that a SCSI command failed with a RESERVATION CONFLICT status.

This symbol has the value 0x800000102.

5.13.3.15 IMA_ERROR_SCSI_STATUS_TASK_SET_FULL

This error indicates that a SCSI command failed with a TASK SET FULL status.

This symbol has the value 0x800000103.

5.13.3.16 IMA_ERROR_SCSI_STATUS_ACA_ACTIVE

This error indicates that a SCSI command failed with a ACA ACTIVE status.

This symbol has the value 0x800000104.

5.13.3.17 IMA_ERROR_SCSI_STATUS_TASK_ABORTED

This error indicates that a SCSI command failed with a TASK ABORTED status.

This symbol has the value 0x800000105.

5.13.3.18 IMA_ERROR_PLUGINS_NOT_SUPPORTED

This error indicates that the library implementation does not support plugins.

This symbol has the value 0x800000106.

5.13.3.19 IMA_ERROR_INVALID_PARAMETER

This error indicates that a specified parameter was invalid.

This error can be returned in a number of situations, such as

- When a client calls an API and specifies a NULL pointer as a parameter to an API that does not accept NULL pointers.
- When a client calls an API and specifies an integer parameter that is out range of the acceptable values for the parameter.
- When a client specifies a pointer to a structure as a parameter to an API and the contents of the structure contain invalid pointers or out of range integer parameters as stated above.

This symbol has the value 0xC0000000.

5.13.3.20 IMA_ERROR_INVALID_OBJECT_TYPE

This error indicates that the object type of the specified [IMA_OID](#) structure is invalid. Most likely an uninitialized variable or a corrupted variable was used in an API call.

This symbol has the value 0xC0000001.

5.13.3.21 IMA_ERROR_INCORRECT_OBJECT_TYPE

This error indicates that an object with an incorrect type was specified in an API call. This can be caused by passing an object ID of the wrong type to an API call. It can also be caused by using an uninitialized variable or a corrupted variable.

This symbol has the value 0xC0000002.

5.13.3.22 IMA_ERROR_OBJECT_NOT_FOUND

This error indicates an object specified in the API call was not found. This can be caused by using an uninitialized variable or a corrupted variable. It can also be caused by using an object ID that referred to an object or plugin that is no longer known to the system.

This symbol has the value 0xC0000003.

5.13.3.23 IMA_ERROR_NAME_TOO_LONG

This error indicates that a name specified in an API call is too long.

This symbol has the value 0xC0000004.

5.13.3.24 IMA_ERROR_UNKNOWN_ERROR

This error indicates that some sort of unknown error has occurred.

This symbol has the value 0x8FFFFFFF.

5.14 IMA_OID

5.14.1 Format

```
typedef struct IMA_oid
{
    IMA_OBJECT_TYPE objectType;
    IMA_UINT32      ownerId;
    IMA_UINT64      objectSequenceNumber;
} IMA_OID;
```

5.14.2 Fields

5.14.2.1 objectType

The type of the object. When an object ID is supplied as a parameter to an API the library uses this value to ensure that the supplied object's type is appropriate for the API that was called.

5.14.2.2 ownerId

A number determined by the library that it uses to uniquely identify the owner of an object. The owner of an object is either the library itself or a plugin. When an object ID is supplied as a parameter to an API the library uses this value to determine if it should handle the call itself or direct the call to one or more plugins.

5.14.2.3 objectSequenceNumber

A number determined by the owner of an object, that is used by the owner possibly in combination with the object type, to uniquely identify an object.

5.14.3 Remarks

This structure shall be treated as opaque by clients of the API. Appropriate APIs, e.g., [IMA_GetObjectType](#) and [IMA_GetAssociatedPluginOid](#), shall be used to extract information from the structure.

5.15 IMA_OID_LIST

5.15.1 Format

```
typedef struct IMA_oid_list
{
    IMA_UINT          oidCount;
    IMA_OID           oids[1];
} IMA_OID_LIST;
```

5.15.2 Fields

5.15.2.1 oidCount

The number of object IDs in the *oids* array.

5.15.2.2 oids

A variable length array of zero or more object IDs. There are *oidCount* object IDs in this array.

5.15.3 Remarks

This structure is used by a number of APIs to return lists of objects. Any instance of this structure returned by an API shall be freed by a client using the [IMA_FreeMemory](#) API.

Although *oids* is declared to be an array of one [IMA_OID](#) structure it can in fact contain any number of [IMA_OID](#) structures.

5.16 IMA_NODE_NAME

5.16.1 Format

```
typedef IMA_WCHAR IMA_NODE_NAME[224];
```

5.16.2 Remarks

This type is used to represent an iSCSI node name. An iSCSI node name is a unique identifier for an iSCSI node.

The name shall be terminated with a Unicode nul character.

Both initiators and targets have node names.

5.17 IMA_NODE_ALIAS

5.17.1 Format

```
typedef IMA_WCHAR IMA_NODE_ALIAS[256];
```

5.17.2 Remarks

This type is used to represent an iSCSI node alias. An iSCSI node alias is intended to be used as a human useful description of an iSCSI node.

The alias shall be terminated with a Unicode nul character.

Both initiators and targets may have node aliases.

5.18 IMA_IP_ADDRESS

5.18.1 Format

```
typedef struct IMA_ip_address
{
    IMA_BOOL            ipv4Address;
    IMA_BYTE            ipAddress[16];

} IMA_IP_ADDRESS;
```

5.18.2 Fields

5.18.2.1 ipv4Address

A boolean indicating the type of IP address this structure represents. If this field has value IMA_TRUE then this is an IPv4 address; if this field has the value IMA_FALSE then this is an IPv6 address.

5.18.2.2 ipAddress

An array of bytes containing the IP address.

- If *ipv4Address* has the value IMA_TRUE then bytes 0 through 3 of this array contain the IP address. Byte 0 of the array contains the most significant byte of the IP address and byte 3 contains the least significant byte of the address.
- If *ipv4Address* has the value IMA_FALSE then bytes 0 through 15 of this array contain the IP address. Byte 0 of the array contains the most significant byte of the IP address and byte 15 contains the least significant byte of the address.

5.19 IMA_HOST_NAME

5.19.1 Format

```
typedef IMA_WCHAR IMA_HOST_NAME[256];
```

5.19.2 Remarks

This type is used to represent a DNS hostname.

This type is used as part of the IMA_HOST_ID union.

The hostname shall be terminated by a Unicode nul character.

5.20 IMA_HOST_ID

5.20.1 Format

```
typedef struct IMA_host_id
{
    IMA_BOOL            hostnameInUse;

    union
    {
        IMA_HOST_NAME    hostname;
        IMA_IP_ADDRESS    ipAddress;
    } id;

} IMA_HOST_ID;
```

5.20.2 Fields

5.20.2.1 `hostnameInUse`

A boolean indicating whether a DNS hostname or IP address is represented by the *IpAddress* field. If this field has the value `IMA_TRUE` then the *hostname* field contains the value. If this field has the value `IMA_FALSE` then the *IpAddress* field contains the value.

5.20.2.2 `hostname`

If *hostnameInUse* has the value `IMA_TRUE` then this field contains a DNS hostname, otherwise the value of this field is undefined.

5.20.2.3 `IpAddress`

If *hostnameInUse* has the value `IMA_FALSE` then this field contains an IP address, otherwise the value of this field is undefined.

5.20.3 Remarks

This type is used to represent a hostname or IP address used in a iSCSI target address.

5.21 IMA_TARGET_ADDRESS

5.21.1 Format

```
typedef struct IMA_target_address
{
    IMA_HOST_ID      hostnameIpAddress;
    IMA_UINT16       portNumber;
} IMA_TARGET_ADDRESS;
```

5.21.2 Fields

5.21.2.1 `hostnameIpAddress`

A DNS hostname or IP address at which a target can be contacted.

5.21.2.2 `portNumber`

The IP port number which can be used in conjunction with the DNS hostname or IP address to contact an iSCSI target.

5.21.3 Remarks

This structure is used to represent a target address that can be used to contact an iSCSI target.

5.22 IMA_ADDRESS_KEY

5.22.1 Format

```
typedef struct IMA_address_key
{
    IMA_IP_ADDRESS  ipAddress;

    IMA_BOOL        portalNumberValid;
    IMA_UINT16      portNumber;

    IMA_BOOL        portalGroupTagValid;
    IMA_UINT16      portalGroupTag;
} IMA_ADDRESS_KEY;
```

5.22.2 Fields

5.22.2.1 `ipAddress`

An IP address at which a target can be contacted.

5.22.2.2 `portNumberValid`

A boolean indicating if the field *portNumber* has a valid value.

5.22.2.3 `portNumber`

The IP port number that is used in conjunction with the IP address to contact an iSCSI target. If *portNumberValid* has the value `IMA_TRUE` then this value is valid. If *portNumberValid* has the value `IMA_FALSE` then this value is undefined.

If *portNumberValid* has the value `IMA_TRUE` then this field shall not be the value zero.

5.22.2.4 `portalGroupTagValid`

A boolean indicating if the field *portalGroupTag* has a valid value.

5.22.2.5 `portalGroupTag`

A 16-bit unsigned integer indicating the portal group tag to be used when connecting to the target. If *portalGroupTagValid* has the value `IMA_TRUE` then this value is valid. If *portalGroupTagValid* has the value `IMA_FALSE` then this value is undefined.

5.22.3 Remarks

This structure is used as a part of the [IMA_ADDRESS_KEYS](#) structure.

5.23 IMA_ADDRESS_KEYS

5.23.1 Format

```
typedef struct IMA_address_keys
{
    IMA_UINT          addressKeyCount;
    IMA_ADDRESS_KEY  addressKeys[1];
} IMA_ADDRESS_KEYS;
```

5.23.2 Fields

5.23.2.1 `addressKeyCount`

The count of address keys in the *addressKeys* array.

5.23.2.2 `addressKeys`

This is an array of one or more target address keys that can be used to contact a target.

5.23.3 Remarks

This structure is returned by the [IMA_GetAddressKeys](#) API.

5.24 IMA_STATIC_DISCOVERY_TARGET

5.24.1 Format

```
typedef struct IMA_static_discovery_target
{
    IMA_NODE_NAME    targetName;
    IMA_TARGET_ADDRESS targetAddress;

    IMA_BOOL          portalGroupTagValid;
    IMA_UINT16        portalGroupTag;
} IMA_STATIC_DISCOVERY_TARGET;
```

5.24.2 Fields

5.24.2.1 targetName

The iSCSI node name of the statically discovered target.

5.24.2.2 targetAddress

A target address at which the statically discovered target resides.

5.24.2.3 portalGroupTagValid

A boolean indicating if the field *portalGroupTag* has a valid value.

5.24.2.4 portalGroupTag

An 16-bit unsigned integer indicating the portal group tag to be used when connecting to the target. If *portalGroupTagValid* has the value IMA_TRUE then this value is valid. If *portalGroupTagValid* has the value IMA_FALSE then this value is undefined.

5.24.3 Remarks

This structure is used to represent a statically discovered target. This structure is used in the [IMA_AddStaticDiscoveryTarget](#) API.

If a valid *portalGroupTag* value is not provided then the initiator should use a SendTargets discovery session to determine a value to use.

5.25 IMA_DISCOVERY_ADDRESS_PROPERTIES

5.25.1 Format

```
typedef struct IMA_discovery_address_properties
{
    IMA_OID          associatedNodeOid;
    IMA_OID          associatedLhbaOid;
    IMA_TARGET_ADDRESS discoveryAddress;
} IMA_DISCOVERY_ADDRESS_PROPERTIES;
```

5.25.2 Fields

5.25.2.1 associatedNodeOid

The object ID of the node through which the target is being accessed.

5.25.2.2 associatedLhbaOid

The object ID of the LHBA that is used to communicate with the discovery address.

5.25.2.3 discoveryAddress

A target address representing the discovery address.

5.25.3 Remarks

This structure is returned by the `IMA_GetDiscoveryAddressProperties` API.

5.26 IMA_STATIC_DISCOVERY_TARGET_PROPERTIES

5.26.1 Format

```
typedef struct IMA_static_discovery_target_properties
{
    IMA_OID          associatedNodeOid;
    IMA_OID          associatedLhbaOid;
    IMA_STATIC_DISCOVERY_TARGET staticConfigTarget;
} IMA_STATIC_DISCOVERY_TARGET_PROPERTIES;
```

5.26.2 Fields

5.26.2.1 associatedNodeOid

The object ID of the node through which the target is being accessed.

5.26.2.2 associatedLhbaOid

The object ID of the LHBA that is used to communicate with the target.

5.26.2.3 staticConfigTarget

The statically configured target.

5.26.3 Remarks

This structure is returned by the `IMA_GetStaticDiscoveryTargetProperties` API.

5.27 IMA_IP_PROPERTIES

5.27.1 Format

```
typedef struct IMA_ip_properties
{
    IMA_BOOL          ipConfigurationMethodSettable;
    IMA_BOOL          dhcpConfigurationEnabled;

    IMA_BOOL          subnetMaskSettable;
    IMA_BOOL          subnetMaskValid;
    IMA_IP_ADDRESS    subnetMask;

    IMA_BOOL          defaultGatewaySettable;
    IMA_BOOL          defaultGatewayValid;
    IMA_IP_ADDRESS    defaultGateway;

    IMA_BOOL          primaryDnsServerAddressSettable;
    IMA_BOOL          primaryDnsServerAddressValid;
    IMA_IP_ADDRESS    primaryDnsServerAddress;

    IMA_BOOL          alternateDnsServerAddressSettable;
    IMA_BOOL          alternateDnsServerAddressValid;
    IMA_IP_ADDRESS    alternateDnsServerAddress;
```

```

        IMA_BYTE          reserved[64];

    } IMA_IP_PROPERTIES;

```

5.27.2 Fields

5.27.2.1 ipConfigurationMethodSettable

A boolean indicating if the IP configuration method is settable using the [IMA_SetIpConfigMethod](#) API. If the value of this field is IMA_TRUE then the [IMA_SetIpConfigMethod](#) API is supported for the associated object. If the value of this field is IMA_FALSE then the API is not supported for that object.

5.27.2.2 dhcpConfigurationEnabled

A boolean indicating if the *subnetMask*, *defaultGateway*, *primaryDnsServerAddress*, and *alternateDnsServerAddress* fields have been set via DHCP or using static configuration. If the value of this field is IMA_TRUE then these fields have been set using DHCP. If the value of this field is IMA_FALSE then these fields have been set via static configuration or they are not otherwise settable.

5.27.2.3 subnetMaskSettable

A boolean indicating if the subnet mask can be set using the [IMA_SetSubnetMask](#) API. If the *dhcpConfigurationEnabled* field has the value IMA_TRUE then this field shall have the value IMA_FALSE. The subnet mask can only be set when static configuration is enabled.

5.27.2.4 subnetMaskValid

A boolean indicating if the field *subnetMask* has a valid value.

5.27.2.5 subnetMask

A structure containing the subnet mask. If *subnetMaskValid* has the value IMA_TRUE then this value is valid. If *subnetMaskValid* has the value IMA_FALSE then this value is undefined.

5.27.2.6 defaultGatewaySettable

A boolean indicating if the default gateway can be set using the [IMA_SetDefaultGateway](#) API. If the *dhcpConfigurationEnabled* field has the value IMA_TRUE then this field shall have the value IMA_FALSE. The default gateway can only be set when static configuration is enabled.

5.27.2.7 defaultGatewayValid

A boolean indicating if the field *defaultGateway* has a valid value.

5.27.2.8 defaultGateway

A structure containing the default gateway. If *defaultGatewayValid* has the value IMA_TRUE then this value is valid. If *defaultGatewayValid* has the value IMA_FALSE then this value is undefined.

5.27.2.9 primaryDnsServerAddressSettable

A boolean indicating if the primary DNS server address can be set using the [IMA_SetDnsServerAddress](#) API. If the *dhcpConfigurationEnabled* field has the value IMA_TRUE then this field shall have the value IMA_FALSE. The primary DNS server can only be set when static configuration is enabled.

5.27.2.10 primaryDnsServerAddressValid

A boolean indicating if the field *primaryDnsServerAddress* has a valid value.

5.27.2.11 primaryDnsServerAddress

An array containing the name or address of the primary DNS server. If *primaryDnsServerAddressValid* has the value IMA_TRUE then this value is valid. If *primaryDnsServerAddressValid* has the value IMA_FALSE then this value is undefined.

5.27.2.12 alternateDnsServerAddressSettable

A boolean indicating if the alternate DNS server can be set using the [IMA_SetDnsServerAddress](#) API. If the *dhcpConfigurationEnabled* field has the value IMA_TRUE then this field shall have the value IMA_FALSE. The alternate DNS server can only be set when static configuration is enabled. If *primaryDnsServerAddressSettable* has the value IMA_FALSE then this field shall have the value IMA_FALSE. The alternate DNS server address can only be set if the primary DNS server can be set.

5.27.2.13 alternateDnsServerAddressValid

A boolean indicating if the field *alternateDnsServerAddress* has a valid value.

5.27.2.14 alternateDnsServerAddress

An array containing the name or address of the alternate DNS server. If *alternateDnsServerAddressValid* has the value IMA_TRUE then this value is valid. If *alternateDnsServerAddressValid* has the value IMA_FALSE then this value is undefined.

5.27.2.15 reserved

This field is reserved.

5.27.3 Remarks

If both the primary and alternate DNS servers are valid then the implementation tries to use the primary DNS server for domain name resolution. If there are errors using the primary DNS server the implementation will use the alternate DNS server. The method that an implementation uses to determine which DNS server to use is implementation specific.

5.28 IMA_LIBRARY_PROPERTIES

5.28.1 Format

```
typedef struct IMA_library_properties
{
    IMA_UINT           supportedImaVersion;
    IMA_WCHAR          vendor[256];
    IMA_WCHAR          implementationVersion[256];
    IMA_WCHAR          fileName[256];
    IMA_DATETIME       buildTime;

    IMA_BYTE           reserved[64];
} IMA_LIBRARY_PROPERTIES;
```

5.28.2 Fields

5.28.2.1 supportedImaVersion

The version of the iSCSI Management API implemented by the library. The value returned by a library for the API as described in this document is one.

5.28.2.2 vendor

A nul terminated Unicode string containing the name of the vendor that created the binary version of the library.

5.28.2.3 implementationVersion

A nul terminated Unicode string containing the implementation version of the library from the vendor specified in *vendor*.

5.28.2.4 fileName

A nul terminated Unicode string ideally containing the path and file name of the library that is being used by the currently executing process can be found. If the path cannot be determined then it is acceptable to fill this field with only the name (and extension if applicable) of the file of the library. If this cannot be determined then this field shall be an empty string.

5.28.2.5 buildTime

The time and date that the library that is executing was built.

5.28.2.6 reserved

This field is reserved.

5.28.3 Remarks

This structure is returned by the [IMA_GetLibraryProperties](#) API.

5.29 IMA_PLUGIN_PROPERTIES

5.29.1 Format

```
typedef struct IMA_plugin_properties
{
    IMA_UINT           supportedImaVersion;
    IMA_WCHAR          vendor[256];
    IMA_WCHAR          implementationVersion[256];
    IMA_WCHAR          fileName[256];
    IMA_DATETIME       buildTime;

    IMA_BOOL           lhasCanBeCreatedAndDestroyed;

    IMA_BYTE           reserved[64];
} IMA_PLUGIN_PROPERTIES;
```

5.29.2 Fields

5.29.2.1 supportedImaVersion

The version of the iSCSI Management API implemented by a plugin. The value returned by a library for the API as described in this document is one.

5.29.2.2 vendor

A nul terminated Unicode string containing the name of the vendor that created the binary version of the plugin.

5.29.2.3 implementationVersion

A nul terminated Unicode string containing the implementation version of the plugin from the vendor specified in *vendor*.

5.29.2.4 fileName

A nul terminated Unicode string ideally containing the path and file name of the plugin that is filing in this structure.

If the path cannot be determined then this field will contain only the name (and extension if applicable) of the file of the plugin. If this cannot be determined then this field will be an empty string.

5.29.2.5 buildTime

The time and date that the plugin that is specified by this structure was built.

5.29.2.6 lhbasCanBeCreatedAndDestroyed

A boolean indicating if LHBAs can be created or destroyed. For this version of the IMA this value is always IMA_FALSE.

5.29.2.7 reserved

This field is reserved.

5.29.3 Remarks

This structure is returned by the [IMA_GetPluginProperties](#) API.

5.30 IMA_NODE_PROPERTIES

5.30.1 Format

```
typedef struct IMA_node_properties
{
    IMA_BOOL        runningInInitiatorMode;
    IMA_BOOL        runningInTargetMode;

    IMA_BOOL        nameValid;
    IMA_NODE_NAME   name;

    IMA_BOOL        aliasValid;
    IMA_NODE_ALIAS   alias;

    IMA_BOOL        nameAndAliasSettable;

    IMA_BYTE        reserved[64];
} IMA_NODE_PROPERTIES;
```

5.30.2 Fields

5.30.2.1 runningInInitiatorMode

A boolean indicating if the node is running as initiator or not.

5.30.2.2 runningInTargetMode

A boolean indicating if the node is running as a target or not.

5.30.2.3 nameValid

A boolean indicating if the node's name is valid or not.

5.30.2.4 name

A nul terminated Unicode string that contains the name of the node.

The value in this field is only valid if *nameValid* is set to IMA_TRUE. If *nameValid* is set to IMA_FALSE then this field will contain an empty string.

5.30.2.5 aliasValid

A boolean indicating if the node's alias is valid or not.

5.30.2.6 alias

A nul terminated Unicode string that contains the alias of the node.

This field is only valid if *aliasValid* is set to IMA_TRUE. If *aliasValid* is set to IMA_FALSE then this field will contain an empty string.

5.30.2.7 nameAndAliasSettable

A boolean indicating if both the name and alias are settable using [IMA_SetNodeName](#) and [IMA_SetNodeAlias](#).

5.30.2.8 reserved

This field is reserved.

5.30.3 Remarks

This structure is returned by the [IMA_GetNodeProperties](#) API.

It is possible for both *runningInInitiatorMode* and *runningInTargetMode* to be set to IMA_TRUE. This means that the node is operating both as an initiator and as a target.

5.31 IMA_LHBA_PROPERTIES**5.31.1 Format**

```
typedef struct IMA_lhba_properties
{
    IMA_WCHAR    osDeviceName[256];
    IMA_BOOL     luExposingSupported;
    IMA_BOOL     isDestroyable;

    IMA_BOOL     staleDataRemovable;
    IMA_UINT     staleDataSize;

    IMA_BOOL     initiatorAuthMethodsSettable;
    IMA_BOOL     targetAuthMethodsSettable;

    IMA_BYTE     reserved[128];
} IMA_LHBA_PROPERTIES;
```

5.31.2 Fields

5.31.2.1 osDeviceName

A nul terminated Unicode string that contains the operating system's name for a logical HBA. If this is an empty string then the device name for the LHBA is unknown.

See Annex A for details on the value(s) of this field for each operating system.

5.31.2.2 luExposingSupported

A boolean indicating if the LHBA supports exposing and unexposing of individual LUs using the [IMA_ExposeLu](#) and [IMA_UnexposeLu](#) APIs. If the value of this field is IMA_TRUE then a client can control the exposing and unexposing of all LUs associated with the LHBA using the [IMA_ExposeLu](#) and [IMA_UnexposeLu](#) APIs. If the value of this field is IMA_FALSE then a client cannot control the exposing and unexposing of LUs using the IMA.

5.31.2.3 isDestroyable

A boolean indicating if the LHBA can be destroyed.

For this version of the IMA this value shall always be IMA_FALSE.

5.31.2.4 staleDataRemovable

A boolean indicating if stale data associated with the LHBA is removable. If this value is IMA_TRUE then the data can be removed using [IMA_RemoveStaleData](#).

5.31.2.5 staleDataSize

The **approximate** size, in bytes, of the amount of stale data associated with the LHBA. If this value is zero then there is no stale data associated with the LHBA.

5.31.2.6 initiatorAuthMethodsSettable

A boolean indicating if the initiator authentication methods used by the LHBA can be set by a client using the [IMA_SetInitiatorAuthMethods](#) API.

5.31.2.7 targetAuthMethodsSettable

A boolean that, for this version of the IMA specification, shall be set to IMA_FALSE.

5.31.2.8 reserved

This field is reserved.

5.31.3 Remarks

This structure is returned by the [IMA_GetLhbaProperties](#) API.

5.32 Upper Level Protocol (ULP) flags

These are flag values used in the *supportedUlps* field of the [IMA_PHBA_PROPERTIES](#) structure.

5.32.1.1 IMA_ULP_TCP

This flag indicates that TCP is supported.

This symbol has the value 0x01.

5.32.1.2 IMA_ULP_SCTP

This flag indicates that SCTP is supported.

This symbol has the value 0x02.

5.32.1.3 IMA_ULP_UDP

This flag indicates that UDP is supported.

This symbol has the value 0x04.

These flags may be OR'd together. For example, it's valid for a PHBA to support both TCP and SCTP. In this case, a value of 0x03 would be returned in the *supportedUlp*s field of the [IMA_PHBA_PROPERTIES](#) structure filled in by a call to [IMA_GetPhbaProperties](#).

5.33 IMA_PHBA_PROPERTIES

5.33.1 Format

```
typedef struct IMA_phba_properties
{
    IMA_WCHAR        vendor[64];
    IMA_WCHAR        model[256];
    IMA_WCHAR        description[256];
    IMA_WCHAR        serialNumber[64];
    IMA_WCHAR        hardwareVersion[256];
    IMA_WCHAR        asicVersion[256];
    IMA_WCHAR        firmwareVersion[256];
    IMA_WCHAR        optionRomVersion[256];
    IMA_WCHAR        driverName[256];
    IMA_WCHAR        driverVersion[256];

    IMA_UINT          supportedUlpS;

    IMA_XBOOL         bidirectionalTransfersSupported;
    IMA_UINT          maximumCdbLength;

    IMA_XBOOL         canBeNic;
    IMA_XBOOL         isNic;

    IMA_XBOOL         isInitiator;
    IMA_XBOOL         isTarget;

    IMA_XBOOL         usingTcpOffloadEngine;
    IMA_XBOOL         usingIscsiOffloadEngine;

    IMA_BYTE          reserved[128];
} IMA_PHBA_PROPERTIES;
```

5.33.2 Fields

5.33.2.1 vendor

A nul terminated Unicode string that contains the name of the vendor of a PHBA. If the first character in this field is nul then the vendor is unknown.

5.33.2.2 model

A nul terminated Unicode string that contains the name of the model of a PHBA. If the first character in this field is nul then the model is unknown.

5.33.2.3 description

A nul terminated Unicode string that contains a description of a PHBA. This is a user friendly description of the PHBA. If the first character in this field is nul then there is no description.

5.33.2.4 serialNumber

A nul terminated Unicode string that contains the serial number of a PHBA. If the first character in this field is nul then the serial number is unknown.

5.33.2.5 hardwareVersion

A nul terminated Unicode string that contains the hardware version of a PHBA. If the first character in this field is nul then the hardware version is unknown.

5.33.2.6 asicVersion

A nul terminated Unicode string that contains the ASIC version of a PHBA. If the first character in this field is nul then the ASIC version is unknown or is not applicable.

5.33.2.7 firmwareVersion

A nul terminated Unicode string that contains the firmware version of a PHBA. If the first character in this field is nul then the firmware version is unknown or is not applicable.

5.33.2.8 optionRomVersion

A nul terminated Unicode string that contains the option ROM version of a PHBA. If the first character in this field is nul then the option ROM version is unknown or is not applicable.

5.33.2.9 driverName

A nul terminated Unicode string that contains the full name of the driver controlling a PHBA. If the first character in this field is nul then the name of the driver is unknown.

On operating systems, such as Windows, where files have extensions the full name includes the extension. The full name does not include the path to the file.

5.33.2.10 driverVersion

A nul terminated Unicode string that contains the version of the driver specified in *driverName*. If the first character in this field is nul then the version of the driver is unknown.

This field can have a known value only if *driverName* has a known value as well.

5.33.2.11 supportedUlp

A field containing flags that indicate what upper level protocols are supported by a PHBA. Examples of upper level protocols include:

- TCP, represented by [IMA_ULP_TCP](#)
- SCTP, represented by [IMA_ULP_SCTP](#)
- UDP, represented by [IMA_ULP_UDP](#)

5.33.2.12 bidirectionalTransfersSupported

A extended boolean that indicates if a PHBA supports executing SCSI commands that cause bidirectional transfers.

NOTE The value of this field applies to the entire “stack”: the hardware, ASIC, firmware, driver, etc. All shall support SCSI commands that cause bidirectional transfers for this field to be set to IMA_TRUE.

5.33.2.13 maximumCdbLength

The maximum length, in bytes, of a CDB that can be transferred by a PHBA. If this field has a value of zero that indicates that this value is unknown.

NOTE The value of this field applies to the entire “stack”: the hardware, ASIC, firmware, driver, etc. All shall support the maximum CDB length returned in this field.

5.33.2.14 canBeNic

An extended boolean that indicates if a PHBA can also function as a “standard” NIC concurrently with functioning as an iSCSI PHBA.

NOTE The value of this field applies to the entire “stack”: the hardware, ASIC, firmware, driver, etc. All shall support the PHBA functioning concurrently as a NIC for the value returned in this field to be IMA_TRUE.

5.33.2.15 isNic

A extended boolean that indicates if a PHBA is functioning as a “standard” NIC concurrently with functioning as an iSCSI PHBA.

5.33.2.16 isInitiator

An extended boolean indicating if the PHBA is functioning as an initiator.

5.33.2.17 isTarget

An extended boolean indicating if the PHBA is functioning as a target.

5.33.2.18 usingTcpOffloadEngine

An extended boolean indicating if the PHBA is using a TCP offload engine.

NOTE This value shall only be set to IMA_TRUE if a TCP offload engine is present and is being used. If it can be determined that a TCP offload engine is present, but it cannot be determined if that offload engine is being used then this value shall be set to IMA_UNKNOWN.

5.33.2.19 usingIscsiOffloadEngine

An extended boolean indicating if the PHBA is using a iSCSI offload engine.

NOTE This value shall only be set to IMA_TRUE if a iSCSI offload engine is present and is being used. If it can be determined that an iSCSI offload engine is present, but it cannot be determined if that offload engine is being used then this value shall be set to IMA_UNKNOWN.

5.33.2.20 reserved

This field is reserved.

5.33.3 Remarks

This structure is returned by the [IMA_GetPhbaProperties](#) API.

Both *isInitiator* and *isTarget* cannot be set to IMA_FALSE as this would mean that the PHBA was not functioning as either an initiator or target: which means that it's not functioning.

5.34 IMA_DISCOVERY_PROPERTIES

5.34.1 Format

```
typedef struct IMA_discovery_properties
{
    IMA_BOOL          iSnsDiscoverySettable;
    IMA_XBOOL         iSnsDiscoveryEnabled;
    IMA_ISNS_DISCOVERY_METHOD iSnsDiscoveryMethod;
    IMA_HOST_ID       iSnsHost;

    IMA_BOOL          slpDiscoverySettable;
    IMA_XBOOL         slpDiscoveryEnabled;

    IMA_BOOL          staticDiscoverySettable;
    IMA_XBOOL         staticDiscoveryEnabled;

    IMA_BOOL          sendTargetsDiscoverySettable;
    IMA_XBOOL         sendTargetsDiscoveryEnabled;

    IMA_BYTE          reserved[128];
} IMA_DISCOVERY_PROPERTIES;
```

5.34.2 Fields

5.34.2.1 iSnsDiscoverySettable

A boolean that indicates if iSNS target discovery can be enabled and disabled using the [IMA_SetIsnsDiscovery](#) API.

5.34.2.2 iSnsDiscoveryEnabled

An extended boolean that indicates if iSNS target discovery is currently enabled or disabled.

NOTE It is possible for this field to have a value of IMA_TRUE and the *iSnsDiscoverySettable* field to have a value of IMA_FALSE. This means that the associated PHBA/LHBA performs iSNS target discovery and that it cannot be disabled.

5.34.2.3 iSnsDiscoveryMethod

A value which indicates how the iSNS server is being discovered. This field is valid only if *iSnsDiscoveryEnabled* has the value IMA_TRUE, otherwise the value of this field is undefined.

5.34.2.4 iSnsHost

A nul terminated Unicode string containing the domain name of the iSNS server the specified PHBA is using. If there is no iSNS server in use, either because iSNS target discovery is disabled or because no iSNS server is found, then this shall be an empty string.

5.34.2.5 slpDiscoverySettable

A boolean that indicates if SLP target discovery can be enabled and disabled using the [IMA_SetSlpDiscovery](#) API.

5.34.2.6 slpDiscoveryEnabled

An extended boolean that indicates if SLP target discovery is currently enabled or disabled.

NOTE It is possible for this field to have a value of IMA_TRUE and the *slpDiscoverySettable* field to have a value of IMA_FALSE. This means that the associated PHBA/LHBA performs SLP target discovery and that it cannot be disabled.

5.34.2.7 **staticDiscoverySettable**

A boolean that indicates if static target discovery can be enabled and disabled using the [IMA_SetStaticDiscovery](#) API.

5.34.2.8 **staticDiscoveryEnabled**

An extended boolean that indicates if static target discovery is currently enabled or disabled.

NOTE It is possible for this field to have a value of IMA_TRUE and the *staticDiscoverySettable* field to have a value of IMA_FALSE. This means that the specified PHBA performs static target discovery and that it cannot be disabled.

5.34.2.9 **sendTargetsDiscoverySettable**

A boolean that indicates if *SendTargets* target discovery can be enabled and disabled using the [IMA_SetSendTargetsDiscovery](#) API.

5.34.2.10 **sendTargetsDiscoveryEnabled**

An extended boolean that indicates if *SendTargets* target discovery is currently enabled or disabled.

NOTE It is possible for this field to have a value of IMA_TRUE and the *sendTargetsDiscoverySettable* field to have a value of IMA_FALSE. This means that the associated PHBA/LHBA performs *SendTargets* target discovery and that it cannot be disabled.

5.34.2.11 **reserved**

This field is reserved.

5.34.3 **Remarks**

This structure is returned by the [IMA_GetDiscoveryProperties](#) API.

It is possible for all of the *xxxDiscoveryEnabled* fields to have the value IMA_FALSE and for all of the *xxxDiscoverySettable* fields to also have the value IMA_FALSE. This indicates that no target discovery is being done by the PHBA/LHBA and no target discovery can be enabled for the PHBA/LHBA. Such a set of discovery properties would indicate that a PHBA/LHBA is acting as a target, but not as an initiator.

5.35 **IMA_PHBA_DOWNLOAD_IMAGE_TYPE**

5.35.1 **Format**

```
typedef enum IMA_phba_download_image_type
{
    IMA_DOWNLOAD_IMAGE_TYPE_FIRMWARE           = 0,
    IMA_DOWNLOAD_IMAGE_TYPE_OPTION_ROM          = 1,
    IMA_DOWNLOAD_IMAGE_TYPE_ALL                  = 2,
} IMA_PHBA_DOWNLOAD_IMAGE_TYPE;
```

5.35.2 **Fields**

5.35.2.1 **IMA_IMAGE_TYPE_FIRMWARE**

This value indicates that a file contains a firmware download image.

This symbol has the value 0.

5.35.2.2 **IMA_IMAGE_TYPE_OPTION_ROM**

This value indicates that a file contains an option ROM download image.