



C++ API Reference

Solstice Enterprise Manager™ 4.1

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 806-7971-10
October 2001, Revision A

Copyright 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, SunSoft, SunExpress, Solstice, Solstice Enterprise Manager, SunOS, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, Californie 94303 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, SunSoft, SunExpress, Solstice, Solstice Enterprise Manager, SunOS, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



Please
Recycle



Adobe PostScript

Contents

Preface xxxvii

- 1. Application Programming Interface 1-1**
 - 1.1 API Classes 1-1
 - 1.2 Viewer API 1-2
 - 1.3 ViewerApi Class 1-2
 - 1.3.1 ViewerApi Member Functions 1-3
 - 1.3.2 Communication Protocol 1-6
 - 1.3.3 ViewerApi Actions 1-6
 - 1.3.4 Event Handling 1-9
 - 1.3.5 Network Views Messages 1-11
 - 1.3.6 Sample Programs 1-15
 - 1.4 Grapher API 1-15
 - 1.5 EMdataset Class 1-16
 - 1.6 EMdynamicDataset Class 1-16
 - 1.6.1 Constructor 1-17
 - 1.6.2 Destructor 1-17
 - 1.7 EMstaticDataset Class 1-17
 - 1.7.1 Constructor 1-17
 - 1.7.2 Destructor 1-18
 - 1.8 EMgraph Class 1-18

1.8.1	Constructor	1-18
1.8.2	Destructor	1-18
1.8.3	EMgraph Member Functions	1-18
1.9	Err Class	1-19
1.9.1	Member Functions	1-20
1.10	Application-to-Application API	1-20
1.11	AppInstComm Class	1-21
1.11.1	Constructors	1-21
1.11.2	Destructor	1-21
1.11.3	AppInstComm Member Functions	1-21
1.12	AppInstObj Class	1-26
1.12.1	Constructors	1-26
1.12.2	Destructor	1-27
1.12.3	AppInstObj Member Functions	1-27
1.13	AppRequest Class	1-28
1.13.1	Constructor	1-28
1.13.2	AppRequest Member Functions	1-28
1.14	Actions	1-29
1.15	Notifications	1-30
1.16	Example	1-31
1.17	AppTarget Class	1-35
1.17.1	Constructor	1-35
1.17.2	AppTarget Member Functions	1-35
2.	Common API	2-1
2.1	Common API Classes	2-1
2.2	Class Categories	2-2
2.3	Variable Types	2-3
2.4	Class Descriptions	2-4
2.5	Address Class	2-5

2.5.1	Constructor	2-6
2.5.2	Operator	2-7
2.5.3	Address Member Functions	2-7
2.6	Arraydeclare Macro	2-8
2.7	Asn1ParsedValue Class	2-9
2.7.1	Constructors	2-9
2.7.2	Asn1ParsedValue Operator Overloading	2-10
2.7.3	Asn1ParsedValue Member Functions	2-10
2.8	Asn1Tag Class	2-11
2.8.1	Constructors	2-12
2.8.2	Asn1Tag Operator Overloading	2-12
2.8.3	Asn1Tag Member Functions	2-13
2.9	Asn1Type Class	2-13
2.9.1	Constructors	2-15
2.9.2	Destructor	2-16
2.9.3	Asn1Type Operator Overloading	2-16
2.9.4	Asn1Type Member Functions	2-17
2.9.5	Related Types	2-27
2.10	Asn1Value Class	2-27
2.10.1	Assignment and Data Sharing	2-28
2.10.2	Type Conversion	2-28
2.10.3	Encoding Functions	2-28
2.10.4	Encoding of a Distinguished Name	2-29
2.10.5	Decoding Simple and Constructed Asn1Values	2-30
2.10.6	Constructors	2-31
2.10.7	Destructor	2-32
2.10.8	Asn1Value Operator Overloading	2-33
2.10.9	Asn1Value Member Functions	2-33
2.10.10	Related Global Functions	2-48
2.11	Blockage Class	2-49

2.11.1	Constructor	2-50
2.11.2	Blockage Member Functions	2-50
2.11.3	Related Global Functions	2-52
2.12	Callback Class	2-55
2.12.1	Constructor	2-56
2.12.2	Callback Operator Overloading	2-56
2.12.3	Callback Member Functions	2-56
2.13	Command Class	2-57
2.13.1	Constructor	2-57
2.13.2	Operator	2-57
2.14	Config Class	2-57
2.14.1	Constructors	2-58
2.14.2	Config Member Functions	2-58
2.15	DataUnit Class	2-59
2.15.1	Constructors	2-60
2.15.2	Destructor	2-62
2.15.3	DataUnit Operator Overloading	2-62
2.15.4	DataUnit Member Functions	2-65
2.16	Dictionary Class	2-69
2.16.1	Constructor	2-69
2.16.2	Dictionary Operator Overloading	2-70
2.16.3	Dictionary Member Functions	2-70
2.17	GenInt Class	2-71
2.17.1	Constructors	2-71
2.17.2	Copy Constructor	2-72
2.17.3	GenInt Member Functions	2-72
2.18	Hash Class	2-79
2.18.1	Hash Member Functions	2-79
2.19	Hashdeclare Macro	2-81
2.20	HashImpl Class	2-81

2.20.1	Constructor	2-83
2.20.2	Destructor	2-83
2.20.3	HashImpl Member Functions	2-83
2.21	Hdict Class	2-85
2.21.1	Constructors	2-86
2.21.2	Hdict (K,T) Operator Overloading	2-86
2.21.3	Hdict Member Functions	2-86
2.22	Hrefdict Class	2-87
2.22.1	Constructors	2-88
2.22.2	Hrefdict (K, T) Operator Overloading	2-88
2.22.3	Hrefdict Member Functions	2-89
2.23	Oid Class	2-90
2.23.1	Constructors	2-91
2.23.2	Oid Operator Overloading	2-91
2.23.3	Oid Member Functions	2-92
2.24	Asn1TypeDefinedType Declarations	2-94
2.24.1	Asn1SubTypeKind	2-95
2.24.2	Asn1SubTypeSize	2-95
2.24.3	Asn1Kind	2-96
2.24.4	Asn1TypeE	2-96
2.24.5	Asn1TypeEL	2-96
2.24.6	Asn1TypeNN	2-97
2.24.7	Asn1TagClass	2-97
2.24.8	Asn1Tagging	2-97
2.25	Queue Class	2-97
2.25.1	Queue Member Functions	2-98
2.26	Queuedeclare Macro	2-100
2.27	Timer Class	2-101
2.27.1	Default Constructor	2-101
2.27.2	Constructor	2-102

2.27.3	Operator	2-102
2.27.4	Related Global Functions	2-102
3.	High-Level PMI	3-1
3.1	Design Objectives	3-1
3.2	Object Management Model	3-2
3.2.1	Naming Objects	3-2
3.2.2	Relationships Between Objects	3-3
3.2.3	Managing Notifications	3-3
3.2.4	Managing Data Types	3-3
3.2.5	Object Schema Management	3-4
3.2.6	Filtering as an Aspect of Album Derivation	3-5
3.3	Meta Data Repository	3-7
3.3.1	getAttribute Action	3-8
3.3.2	getAllDocuments Action	3-8
3.3.3	getAsn1Module Action	3-8
3.3.4	getObjectClass Action	3-8
3.3.5	getDocument Action	3-9
3.3.6	getPackage Action	3-9
3.3.7	getPackagesByOC Action	3-14
3.3.8	getOidName Action	3-16
3.3.9	Sample MDR Action Program	3-17
3.4	Symbolic Constants	3-18
3.5	Defined Types	3-21
3.5.1	Asn1Int	3-22
3.5.2	CCB	3-22
3.5.3	CDU	3-22
3.5.4	DU	3-22
3.5.5	FBits	3-22
3.6	Error Handling and Event Dispatching	3-24

3.6.1	Event Dispatching Functions	3-25
3.7	pmi_sched_get_fds Function	3-25
3.8	High-Level PMI Classes	3-26
3.9	Album Class	3-27
3.9.1	Constructors	3-29
3.9.2	Album Operator Overloading	3-30
3.9.3	Album Member Functions	3-30
3.10	AlbumImage Class	3-52
3.10.1	Constructors	3-53
3.10.2	Destructors	3-54
3.10.3	AlbumImage Operator Overloading	3-54
3.10.4	AlbumImage Member Functions	3-55
3.11	AppTarget Class	3-56
3.11.1	Constructors	3-56
3.11.2	AppTarget Operator Overloading	3-56
3.12	AuthApps Class	3-56
3.12.1	Constructors	3-57
3.12.2	AuthApps Operator Overloading	3-57
3.12.3	AuthApps Member Functions	3-57
3.13	AuthFeatures Class	3-58
3.13.1	Constructor	3-59
3.13.2	AuthFeatures Operator Overloading	3-59
3.13.3	AuthFeatures Member Functions	3-59
3.14	Coder Class	3-60
3.14.1	Constructors	3-61
3.14.2	Coder Operator Overloading	3-61
3.14.3	Coder Member Functions	3-62
3.15	CurrentEvent Class	3-62
3.15.1	Constructors	3-64
3.15.2	CurrentEvent Operator Overloading	3-64

- 3.15.3 `CurrentEvent` Member Functions 3-65
- 3.16 `Error` Class 3-72
 - 3.16.1 `Constructor` 3-73
 - 3.16.2 `Error` Operator Overloading 3-73
 - 3.16.3 `Error` Public Data Member 3-73
 - 3.16.4 `Error` Member Functions 3-74
 - 3.16.5 `Error` Types and Strings 3-76
- 3.17 `Image` Class 3-77
 - 3.17.1 `Image` Constructor 3-80
 - 3.17.2 `Image` Operator Overloading 3-81
 - 3.17.3 `Image` Member Functions 3-81
 - 3.17.4 Related Global Functions 3-115
- 3.18 `Morf` Class 3-116
 - 3.18.1 `Constructors` 3-118
 - 3.18.2 `Destructor` 3-120
 - 3.18.3 `Morf` Operator Overloading 3-120
 - 3.18.4 `Morf` Member Functions 3-121
- 3.19 `MorfBuilder` Class 3-135
 - 3.19.1 `Constructors` 3-136
 - 3.19.2 `Destructor` 3-137
 - 3.19.3 `MorfBuilder` Operator Overloading 3-137
 - 3.19.4 `MorfBuilder` Member Functions 3-137
- 3.20 `PasswordTty` Class 3-148
 - 3.20.1 `Constructors` 3-149
 - 3.20.2 `PasswordTty` Operator Overloading 3-149
 - 3.20.3 `PasswordTty` Member function 3-149
- 3.21 `Platform` Class 3-149
 - 3.21.1 `Constructors` 3-151
 - 3.21.2 `Destructor` 3-151
 - 3.21.3 `Platform` Operator Overloading 3-151

3.21.4	Platform Member Functions	3-152
3.21.5	GETENV Macro	3-168
3.22	Syntax Class	3-168
3.22.1	Constructors	3-169
3.22.2	Syntax Operator Overloading	3-170
3.22.3	Syntax Member Functions	3-171
3.23	Waiter Class	3-175
3.23.1	Constructors	3-176
3.23.2	Waiter Operator Overloading	3-179
3.23.3	Waiter Member Functions	3-179
4.	Low-Level PMI	4-1
4.1	Communication Path	4-1
4.2	Root Classes for the Low-Level PMI	4-3
4.3	Low-Level PMI Classes	4-3
4.3.1	Class Inheritance	4-3
4.3.2	Class Summary	4-6
4.4	AccessDenied Class	4-9
4.4.1	Constructor	4-9
4.5	ActionReq Class	4-10
4.5.1	Constructor	4-10
4.6	ActionRes Class	4-11
4.6.1	Constructor	4-11
4.7	AssocReleased Class	4-12
4.7.1	Constructor	4-12
4.8	CancelGetReq Class	4-13
4.8.1	Constructor	4-13
4.9	CancelGetRes Class	4-14
4.9.1	Constructor	4-14
4.10	ClassInstConfl Class	4-15

4.10.1	Constructor	4-15
4.11	CreateReq Class	4-16
4.11.1	Constructor	4-16
4.12	CreateRes Class	4-17
4.12.1	Constructor	4-17
4.13	DeleteReq Class	4-18
4.13.1	Constructor	4-18
4.14	DeleteRes Class	4-19
4.14.1	Constructor	4-19
4.15	DuplicateOI Class	4-20
4.15.1	Constructor	4-20
4.16	DupMessageId Class	4-21
4.16.1	Constructor	4-21
4.17	ErrorResUnexp Class	4-22
4.17.1	Constructor	4-22
4.18	EventReq Class	4-23
4.18.1	Constructor	4-23
4.19	GetListErr Class	4-24
4.19.1	Constructor	4-24
4.20	GetReq Class	4-25
4.20.1	Constructor	4-25
4.21	GetRes Class	4-26
4.21.1	Constructor	4-26
4.22	InvalidActionArg Class	4-27
4.22.1	Constructor	4-27
4.23	InvalidAttrVal Class	4-28
4.23.1	Constructor	4-28
4.24	InvalidEventArg Class	4-29
4.24.1	Constructor	4-29
4.25	InvalidFilter Class	4-30

4.25.1	Constructor	4-30
4.26	InvalidOI Class	4-31
4.26.1	Constructor	4-31
4.27	InvalidOperation Class	4-32
4.27.1	Constructor	4-32
4.28	InvalidOperator Class	4-33
4.28.1	Constructor	4-33
4.29	InvalidScope Class	4-34
4.29.1	Constructor	4-34
4.30	LinkedResUnexp Class	4-35
4.30.1	Constructor	4-35
4.31	Message Class	4-36
4.31.1	Constructor	4-37
4.31.2	Message Member Functions	4-37
4.32	MessageSAP Class	4-39
4.32.1	Constructor	4-41
4.32.2	MessageSAP Member Functions	4-41
4.32.3	MessageSAP Initialization	4-44
4.33	MessQOS Class	4-45
4.34	MessScope Class	4-45
4.34.1	Constructors	4-46
4.35	MissingAttrVal Class	4-47
4.35.1	Constructor	4-47
4.36	MistypedArg Class	4-48
4.36.1	Constructor	4-48
4.37	MistypedError Class	4-49
4.37.1	Constructor	4-49
4.38	MistypedOp Class	4-50
4.38.1	Constructor	4-50
4.39	MistypedRes Class	4-51

4.39.1	Constructor	4-51
4.40	NoSuchAction Class	4-52
4.40.1	Constructor	4-52
4.41	NoSuchActionArg Class	4-53
4.41.1	Constructor	4-53
4.42	NoSuchAttr Class	4-54
4.42.1	Constructor	4-54
4.43	NoSuchEvent Class	4-55
4.43.1	Constructor	4-55
4.44	NoSuchEventArgs Class	4-56
4.44.1	Constructor	4-56
4.45	NoSuchMessageId Class	4-57
4.45.1	Constructor	4-57
4.46	NoSuchOC Class	4-58
4.46.1	Constructor	4-58
4.47	NoSuchOI Class	4-59
4.47.1	Constructor	4-59
4.48	NoSuchRefOI Class	4-60
4.48.1	Constructor	4-60
4.49	ObjReqMess Class	4-61
4.49.1	Constructor	4-61
4.50	ObjResMess Class	4-62
4.50.1	Constructor	4-62
4.51	OpCancelled Class	4-63
4.51.1	Constructor	4-63
4.52	ProcessFailure Class	4-64
4.52.1	Constructor	4-64
4.53	ReqMess Class	4-65
4.53.1	Constructor	4-65
4.54	ResMess Class	4-66

4.54.1	Constructor	4-66
4.55	ResourceLimit Class	4-67
4.55.1	Constructor	4-67
4.56	ScopedReqMess Class	4-68
4.56.1	Constructor	4-68
4.57	SetListErr Class	4-69
4.57.1	Constructor	4-69
4.58	SetReq Class	4-70
4.58.1	Constructor	4-70
4.59	SetRes Class	4-71
4.59.1	Constructor	4-71
4.60	SyncNotSupp Class	4-72
4.60.1	Constructor	4-72
4.61	TimedOut Class	4-73
4.61.1	Constructor	4-73
4.62	UnexpChildOp Class	4-74
4.62.1	Constructor	4-74
4.63	UnexpError Class	4-75
4.63.1	Constructor	4-75
4.64	UnexpRes Class	4-76
4.64.1	Constructor	4-76
4.65	UnrecError Class	4-77
4.65.1	Constructor	4-77
4.66	UnrecLinkId Class	4-78
4.66.1	Constructor	4-78
4.67	UnrecMessageId Class	4-79
4.67.1	Constructor	4-79
4.68	UnrecOp Class	4-80
4.68.1	Constructor	4-80
4.69	Constants and Defined Types	4-81

- 4.69.1 `MessId` 4-81
- 4.69.2 `MessMode` 4-82
- 4.69.3 `MessagePtr` 4-82
- 4.69.4 `MessScopeType` 4-82
- 4.69.5 `MessSync` 4-83
- 4.69.6 `MessBaseType` 4-83
- 4.69.7 `MessType` 4-84
- 4.69.8 `MESSTYPE_MAX` 4-86
- 4.69.9 `ResponseHandle` 4-86
- 4.69.10 `SendResult` 4-86

5. Access Control API 5-1

- 5.1 Design Objectives 5-1
- 5.2 Access Control Types 5-2
- 5.3 Class Hierarchy 5-2
- 5.4 Symbolic Constants and Defined Types 5-4
 - 5.4.1 Constants 5-4
 - 5.4.2 Defined Types 5-8
- 5.5 Access Control API Classes 5-11
- 5.6 `ACAccessControlRules` Class 5-12
 - 5.6.1 Constructor 5-12
 - 5.6.2 Destructor 5-13
 - 5.6.3 `ACAccessControlRules` Member Functions 5-13
- 5.7 `ACAccessUserList` Class 5-20
 - 5.7.1 Constructor 5-20
 - 5.7.2 Destructor 5-20
 - 5.7.3 `ACAccessUserList` Member Functions 5-21
- 5.8 `ACAppFeatureContainer` Class 5-23
 - 5.8.1 Constructor 5-23
 - 5.8.2 Destructor 5-23

5.8.3	ACAppFeatureContainer Member Functions	5-23
5.9	ACApplication Class	5-24
5.9.1	Constructor	5-24
5.9.2	Destructor	5-25
5.9.3	ACApplication Member Functions	5-25
5.10	ACApplicationContainer Class	5-26
5.10.1	Constructor	5-26
5.10.2	Destructor	5-26
5.10.3	ACApplicationContainer Member Functions	5-26
5.11	ACApplicationFeature Class	5-27
5.11.1	Constructor	5-27
5.11.2	Destructor	5-28
5.11.3	ACApplicationFeature Member Functions	5-28
5.12	ACCallback Class	5-29
5.12.1	Constructors	5-29
5.12.2	Destructor	5-30
5.12.3	ACCallback Operator Overloading	5-30
5.12.4	ACCallback Member Functions	5-30
5.13	ACContainer Class	5-31
5.13.1	Constructor	5-31
5.13.2	Destructor	5-31
5.13.3	ACContainer Operator Overloading	5-32
5.13.4	ACContainer Member Functions	5-32
5.14	ACDBObject Class	5-35
5.14.1	Constructor	5-35
5.14.2	Destructor	5-35
5.14.3	ACDBObject Member Functions	5-36
5.14.4	Notes About the ACDBObject Class	5-38
5.15	ACDBObjectContainer Class	5-40
5.15.1	Constructor	5-40

- 5.15.2 Destructor 5-40
 - 5.15.3 ACDBObjectContainer Member Functions 5-40
- 5.16 ACEMNotificationEmitter Class 5-42
 - 5.16.1 Constructor 5-42
 - 5.16.2 Destructor 5-42
 - 5.16.3 ACEMNotificationEmitter Member Functions 5-43
- 5.17 ACETargets Class 5-44
 - 5.17.1 Constructor 5-45
 - 5.17.2 Destructor 5-45
 - 5.17.3 ACETargets Member Functions 5-45
- 5.18 ACGroup Class 5-46
 - 5.18.1 Constructor 5-46
 - 5.18.2 Destructor 5-46
 - 5.18.3 ACGroup Member Functions 5-47
- 5.19 ACGroupContainer Class 5-52
 - 5.19.1 Constructor 5-53
 - 5.19.2 Destructor 5-53
 - 5.19.3 ACGroupContainer Member Functions 5-53
- 5.20 ACInterface Class 5-54
 - 5.20.1 Constructor 5-54
 - 5.20.2 Destructor 5-55
 - 5.20.3 ACInterface Member Functions 5-55
- 5.21 ACOBJECT Class 5-57
 - 5.21.1 Constructor 5-57
 - 5.21.2 Destructor 5-57
 - 5.21.3 ACOBJECT Operator Overloading 5-58
 - 5.21.4 ACOBJECT Member Functions 5-58
- 5.22 ACRule Class 5-62
 - 5.22.1 Constructor 5-63
 - 5.22.2 Destructor 5-63

5.22.3	ACRule Member Functions	5-63
5.23	ACRuleContainer Class	5-66
5.23.1	Constructor	5-66
5.23.2	Destructor	5-67
5.23.3	ACRuleContainer Member Functions	5-67
5.24	ACScope Class	5-68
5.24.1	Constructors	5-68
5.25	ACTargets Class	5-69
5.25.1	Constructor	5-70
5.25.2	Destructor	5-70
5.25.3	ACTargets Member Functions	5-70
5.26	ACTargetsContainer Class	5-74
5.26.1	Constructor	5-74
5.26.2	Destructor	5-75
5.26.3	ACTargetsContainer Member Functions	5-75
5.27	ACUser Class	5-76
5.27.1	Constructors	5-76
5.27.2	ACUser Operator Overloading	5-77
5.27.3	ACUser Member Functions	5-77
6.	Access Control Engine API	6-1
6.1	Symbolic Constants	6-2
6.1.1	ACEOperationType	6-2
6.1.2	ACEEnforcementAction	6-2
6.2	ACE API Classes	6-3
6.3	ACE Class	6-3
6.3.1	Constructor	6-4
6.3.2	Destructor	6-4
6.3.3	ACE Member Functions	6-4
6.4	ACEContext Class	6-6

- 6.4.1 Constructor 6-7
 - 6.4.2 Destructor 6-7
 - 6.4.3 ACEContext Operator Overloading 6-7
 - 6.4.4 ACEContext Member Functions 6-7
- 6.5 ACEDecision Class 6-9
 - 6.5.1 Constructor 6-9
 - 6.5.2 Destructor 6-10
 - 6.5.3 ACEDecision Member Functions 6-10
- 6.6 ACEDomain Class 6-11
 - 6.6.1 Constructor 6-11
 - 6.6.2 Destructor 6-11
 - 6.6.3 ACEDomain Member Function 6-11
- 6.7 ACEReqData Class 6-12
 - 6.7.1 Constructor 6-12
 - 6.7.2 Destructor 6-13
- 6.8 AuxServerUtils Class 6-14
 - 6.8.1 Constructor 6-14
 - 6.8.2 Destructor 6-14
 - 6.8.3 AuxServerUtils Virtual Functions 6-15

7. Nerve Center Interface 7-1

- 7.1 Requests 7-1
- 7.2 Class and Function Summary 7-2
- 7.3 NC Requests 7-3
 - 7.3.1 Synchronous Launches 7-3
 - 7.3.2 Asynchronous Launches 7-4
- 7.4 NCI Library Classes 7-5
- 7.5 NCAsyncResIterator Class 7-5
 - 7.5.1 Constructor 7-5
 - 7.5.2 Destructor 7-6

7.5.3	Operator Overloading for Prefix Operator++	7-6
7.5.4	Member Functions	7-6
7.6	NCParsedReqHandle Class	7-7
7.6.1	Constructors	7-8
7.6.2	Default Destructor	7-8
7.6.3	Member Functions	7-8
7.7	NCTopoInfoList Class	7-9
7.7.1	Default Constructor	7-10
7.7.2	Copy Constructor	7-10
7.7.3	Destructor	7-10
7.7.4	Operator Overloading for Operator=	7-10
7.7.5	Member Functions	7-11
7.8	NCI Library Functions	7-11
7.9	NCI Global Variables	7-13
7.9.1	nci_error_reason	7-13
7.9.2	topoNodeId Argument	7-13
7.10	NCI Functions	7-14
7.10.1	nci_action_add	7-14
7.10.2	nci_action_delete	7-14
7.10.3	nci_async_request_start	7-15
7.10.4	nci_condition_add	7-16
7.10.5	nci_condition_delete	7-16
7.10.6	nci_condition_get	7-16
7.10.7	nci_init	7-17
7.10.8	nci_parse_handle	7-18
7.10.9	nci_pollrate_add	7-18
7.10.10	nci_pollrate_delete	7-19
7.10.11	nci_request_delete	7-19
7.10.12	nci_request_dump	7-19
7.10.13	nci_request_info	7-20

7.10.14	nci_request_list	7-20
7.10.15	nci_request_start	7-21
7.10.16	nci_severity_add	7-23
7.10.17	nci_severity_delete	7-24
7.10.18	nci_state_add	7-24
7.10.19	nci_state_delete	7-24
7.10.20	nci_state_get	7-25
7.10.21	nci_template_add	7-25
7.10.22	nci_template_copy	7-26
7.10.23	nci_template_create	7-26
7.10.24	nci_template_delete	7-26
7.10.25	nci_template_find	7-27
7.10.26	nci_template_revert	7-27
7.10.27	nci_template_store	7-28
7.10.28	nci_transition_add	7-28
7.10.29	nci_transition_delete	7-29
7.10.30	nci_transition_find	7-29
7.10.31	nci_transition_get	7-30

8. Topology API 8-1

8.1	Topology Classes	8-2
8.1.1	General Comments	8-3
8.1.2	General Description	8-3
8.2	Class Overview	8-5
8.2.1	Relationship to the GDMO	8-5
8.2.2	Relationship to PMI	8-6
8.3	EMTopoPlatform Class	8-7
8.3.1	get_attributes_by_mo()	8-8
8.3.2	set_attributes_by_mo()	8-9
8.4	Persistent Object Classes	8-10

8.4.1	EMObject Class	8-10
8.4.2	EMObject Member Functions	8-11
8.4.3	EMTopoType Class	8-11
8.4.4	EMTopoNode Class	8-12
8.4.5	EMSnmpAgent Class	8-12
8.4.6	EMCmipAgent Class	8-12
8.4.7	EMRpcAgent Class	8-13
8.5	Utility Classes	8-13
8.5.1	EMIntegerSet Class	8-13
8.5.2	EMStatus Class	8-13
8.6	Topology API Concepts	8-14
8.6.1	Element Naming	8-14
8.6.2	Duplicate Topology Node Names	8-14
8.6.3	MIS-MIS Awareness	8-15
8.6.4	Performance Considerations	8-15
8.7	Examples	8-15
8.7.1	Makefile	8-15
8.7.2	Finding Topology Nodes	8-16
8.7.3	Registering Events for EMTopoNode	8-20
8.7.4	Printing the Topology Hierarchy	8-25
8.8	Class Reference	8-29
8.9	EMStatus Class	8-29
8.9.1	Constructors and Destructor	8-31
8.9.2	Operators	8-31
8.9.3	Global Operators	8-32
8.10	EMIntegerSet Class	8-32
8.10.1	Example	8-33
8.10.2	Constructors and Destructor	8-33
8.10.3	Operators	8-34
8.10.4	Member Functions	8-35

- 8.10.5 Global Operators 8-37
- 8.11 EMIntegerSetIterator Class 8-37
 - 8.11.1 Example 8-38
 - 8.11.2 Constructors and Destructor 8-38
 - 8.11.3 Member Functions 8-39
- 8.12 EMTopoPlatform Class 8-39
 - 8.12.1 Example 8-40
 - 8.12.2 Static Member Functions 8-41
 - 8.12.3 Access Member Functions 8-42
 - 8.12.4 General Member Functions 8-43
- 8.13 EMObject Class 8-47
 - 8.13.1 Constructors and Destructor 8-48
 - 8.13.2 EMObject Member Functions Supported By POC Classes 8-49
 - 8.13.3 Operators Supported by all POC classes 8-52
 - 8.13.4 Other Member Functions Supported by POC Classes. 8-53
 - 8.13.5 Static Member Functions Supported by POC Classes 8-54
- 8.14 EMTopoNodeDn Class 8-55
 - 8.14.1 Constructors and Destructor 8-56
 - 8.14.2 Operators 8-57
 - 8.14.3 Access Member Functions 8-57
 - 8.14.4 General Member Functions 8-58
 - 8.14.5 Related Global Operators 8-58
- 8.15 EMTopoNode Class 8-59
 - 8.15.1 Example 8-62
 - 8.15.2 Constructors and Destructor 8-68
 - 8.15.3 Access Member Functions 8-68
 - 8.15.4 Static Member Functions for Event Subscription 8-91
 - 8.15.5 Related Global Operators 8-92
- 8.16 EMTopoTypeDn Class 8-93
 - 8.16.1 Constants 8-93

8.16.2	Constructors and Destructor	8-93
8.16.3	Operators	8-94
8.16.4	Access Member Functions	8-95
8.16.5	General Member Functions	8-95
8.17	EMTopoType Class	8-96
8.17.1	Example	8-97
8.17.2	Constructors and Destructor	8-98
8.17.3	Operators	8-99
8.17.4	Access Member Functions	8-99
8.17.5	Static Member Functions	8-103
8.17.6	Static Member Functions for Event Subscription	8-105
8.17.7	Global Operators	8-106
8.18	EMAgent Class	8-107
8.18.1	Access Member Functions	8-108
8.19	EMCmipAgentDn Class	8-109
8.19.1	Constructors and Destructor	8-109
8.19.2	Operators	8-110
8.19.3	Access Member Functions	8-110
8.19.4	General Member Functions	8-111
8.19.5	Related Global Operators	8-111
8.20	EMCmipAgent Class	8-112
8.20.1	Example	8-114
8.20.2	Access Member Functions	8-115
8.20.3	Global Operators	8-123
8.21	EMRpcAgentDn Class	8-123
8.21.1	Constructors and Destructor	8-123
8.21.2	Operators	8-124
8.21.3	Access Member Functions	8-125
8.21.4	General Member Functions	8-125
8.21.5	Global Operators	8-126

- 8.22 `EMRpcAgent` Class 8-126
 - 8.22.1 Example 8-127
 - 8.22.2 Constructors and Destructor 8-129
 - 8.22.3 Access Member Functions 8-129
 - 8.22.4 Global Operators 8-132
- 8.23 `EMSnmpAgentDn` Class 8-132
 - 8.23.1 Constructors, and Destructor 8-133
 - 8.23.2 Operators 8-133
 - 8.23.3 Access Member Functions 8-134
 - 8.23.4 General Member Functions 8-134
 - 8.23.5 Global Operators 8-135
- 8.24 `EMSnmpAgent` Class 8-135
 - 8.24.1 Example 8-137
 - 8.24.2 Constructors and Destructor 8-139
 - 8.24.3 Access Member Functions 8-140
 - 8.24.4 Related Global Operators 8-145

9. Object Services API 9-1

- 9.1 Operational Flow 9-2
- 9.2 Service Request Function Parameters 9-3
- 9.3 Service Response Callback Function Parameters 9-7
- 9.4 Services Interface Descriptions and Examples 9-7
 - 9.4.1 Get Request Service 9-8
 - 9.4.2 Get Response Callback 9-15
 - 9.4.3 Set Request Service 9-19
 - 9.4.4 Set Response Callback 9-24
 - 9.4.5 Action Request Service 9-26
 - 9.4.6 Action Response Callback 9-30
 - 9.4.7 Create Request Service 9-33
 - 9.4.8 Create Response Callback 9-39

9.4.9	Delete Request Service	9-42
9.4.10	Delete Response Callback	9-49
9.4.11	Delete Response Callback Parameter Description	9-49
9.4.12	Event Report Request Service (Unconfirmed)	9-53
9.4.13	Event Report Response Callback	9-57
9.5	Supporting Functions for Example Code	9-57
9.5.1	Debugging Flags	9-58
9.5.2	get_sys_dn Function	9-58
9.5.3	get_graphstr_rdn Function	9-59

Figures

FIGURE 4-1	Applications to MRM Communication	4-2
FIGURE 4-2	Inheritance Tree of the <code>Message</code> Class	4-5
FIGURE 4-3	Inheritance Tree of the <code>MessageSAP</code> Class	4-40
FIGURE 5-1	C++ Container Classes and Their Inheritance	5-2
FIGURE 5-2	Access Control C++ Classes and Their Inheritance	5-3
FIGURE 8-1	Position of the Topology API	8-4

Tables

TABLE 1-1	Application Programming Interface Classes	1-1
TABLE 1-2	ViewerApi Class Defined Actions	1-6
TABLE 1-3	Network Views Event Messages	1-10
TABLE 1-4	Using ViewRegisterEvents	1-11
TABLE 2-1	Common API Classes	2-1
TABLE 2-2	Class Categories	2-2
TABLE 2-3	Basic Variable Types	2-3
TABLE 2-4	Common API Classes	2-4
TABLE 2-5	AddressClass Data Members	2-5
TABLE 2-6	AddressTag Data Members	2-5
TABLE 2-7	Directory Services	2-6
TABLE 2-8	Address Public Variables	2-6
TABLE 2-9	Asn1ParsedValue Public Functions	2-9
TABLE 2-10	Asn1Tag Public Variables	2-12
TABLE 2-11	Asn1Tag Public Functions	2-12
TABLE 2-12	Asn1Type Public Functions	2-14
TABLE 2-13	Asn1Value Functions	2-30
TABLE 2-14	decode_ext Variable Descriptions	2-36
TABLE 2-15	encode_ext Arguments	2-41

TABLE 2-16	Blockage Public Functions	2-49
TABLE 2-17	Callback Public Variables	2-55
TABLE 2-18	Callback Public Functions	2-55
TABLE 2-19	Config Public Functions	2-58
TABLE 2-20	DataUnit Public Functions	2-60
TABLE 2-21	Dictionary Protected Variables	2-69
TABLE 2-22	Dictionary Functions	2-69
TABLE 2-23	HashImpl Public Functions	2-82
TABLE 2-24	Hdict Protected Variables	2-85
TABLE 2-25	Hdict Public Functions	2-85
TABLE 2-26	Hrefdict Protected Variables	2-88
TABLE 2-27	Hrefdict Public Functions	2-88
TABLE 2-28	Oid Public Functions	2-90
TABLE 3-1	Properties in Album, Image, and Platform	3-4
TABLE 3-2	Scoping Parameters	3-6
TABLE 3-3	String Constants	3-19
TABLE 3-4	Format Bit Values on <code>get</code> Function Calls	3-22
TABLE 3-5	Format Bit Values on <code>set</code> Function Calls	3-23
TABLE 3-6	High-Level PMI Classes	3-26
TABLE 3-7	Album Method Types	3-27
TABLE 3-8	Properties Supported by Most Albums	3-42
TABLE 3-9	Events Supported by Album	3-52
TABLE 3-10	CurrentEvent Method Types	3-63
TABLE 3-11	Error Types	3-76
TABLE 3-12	Image Method Types	3-78
TABLE 3-13	Properties Supported by Most Image Attributes	3-92
TABLE 3-14	Properties Supported by Most Images	3-98
TABLE 3-15	Image-specific Asynchronous Events	3-115

TABLE 3-16	Morf Method Types	3-117
TABLE 3-17	Properties Addressed by <i>key</i>	3-144
TABLE 3-18	Platform Method Types	3-150
TABLE 3-19	MIS Properties Supported	3-163
TABLE 3-20	MIS Events Supported	3-167
TABLE 3-21	Syntax Method Types	3-169
TABLE 3-22	Waiter Method Types	3-175
TABLE 4-1	Low-Level PMI Classes	4-6
TABLE 4-2	AccessDenied Public Data Member	4-9
TABLE 4-3	ActionReq Public Data Members	4-10
TABLE 4-4	ActionRes Public Data Members	4-11
TABLE 4-5	CancelGetReq Public Variable	4-13
TABLE 4-6	ClassInstConfl Public Data Members	4-15
TABLE 4-7	CreateReq Public Data Members	4-16
TABLE 4-8	CreateRes Public Data Members	4-17
TABLE 4-9	DeleteRes Public Data Member	4-19
TABLE 4-10	DuplicateOI Public Variable	4-20
TABLE 4-11	EventReq Public Data Members	4-23
TABLE 4-12	GetListErr Public Data Members	4-24
TABLE 4-13	GetReq Public Variable	4-25
TABLE 4-14	GetRes Public Data Members	4-26
TABLE 4-15	InvalidActionArg Public Data Members	4-27
TABLE 4-16	InvalidAttrVal Public Variable	4-28
TABLE 4-17	InvalidEventArg Public Data Members	4-29
TABLE 4-18	InvalidFilter Public Variable	4-30
TABLE 4-19	InvalidOI Public Variable	4-31
TABLE 4-20	InvalidOperator Public Variable	4-33
TABLE 4-21	InvalidScope Public Variable	4-34

TABLE 4-22	Message Class Public Data Members	4-36
TABLE 4-23	Message Class Method Types	4-36
TABLE 4-24	MessageSAP Subclasses	4-39
TABLE 4-25	MessageSAP Public Data Members	4-40
TABLE 4-26	MessageSAP Method Types	4-40
TABLE 4-27	Types of MessScope Scoping	4-45
TABLE 4-28	MessScope Public Data Members	4-46
TABLE 4-29	MissingAttrVal Public Variable	4-47
TABLE 4-30	NoSuchAction Public Data Members	4-52
TABLE 4-31	NoSuchActionArg Public Data Members	4-53
TABLE 4-32	NoSuchAttr Public Variable	4-54
TABLE 4-33	NoSuchEvent Public Data Members	4-55
TABLE 4-34	NoSuchEventArgs Public Data Members	4-56
TABLE 4-35	NoSuchMessageId Public Variable	4-57
TABLE 4-36	NoSuchOC Public Variable	4-58
TABLE 4-37	NoSuchOI Public Variable	4-59
TABLE 4-38	NoSuchRefOI Public Variable	4-60
TABLE 4-39	ObjReqMess Public Data Members	4-61
TABLE 4-40	ObjResMess Public Data Members	4-62
TABLE 4-41	ProcessFailure Public Variable	4-64
TABLE 4-42	ReqMess Public Data Members	4-65
TABLE 4-43	ResMess Public Variable	4-66
TABLE 4-44	ScopedReqMess Public Data Members	4-68
TABLE 4-45	SetListErr Public Data Members	4-69
TABLE 4-46	SetReq Public Variable	4-70
TABLE 4-47	SetRes Public Data Members	4-71
TABLE 4-48	SyncNotSupp Public Variable	4-72
TABLE 5-1	Access Control API Classes	5-3

TABLE 6-1	<code>check_access()</code> Parameters	6-5
TABLE 6-2	<code>ACEReqData()</code> Constructor Parameters	6-12
TABLE 7-1	Nerve Center Classes and Functions	7-2
TABLE 7-2	NCI Library Classes	7-5
TABLE 7-3	Nerve Center Library Functions	7-12
TABLE 8-1	Topology API Classes	8-2
TABLE 8-2	Topology API and GDMO Object Relationship	8-5
TABLE 8-3	<i>cache_view_graph</i> Option	8-42
TABLE 8-4	EMTopoNode Attributes Table	8-59
TABLE 8-5	EM TopoType Attributes	8-96
TABLE 8-6	EMCmipAgent Attributes	8-112
TABLE 8-7	EMRpcAgent Attributes	8-127
TABLE 8-8	EMSnmppAgent Attributes	8-136
TABLE 9-1	Service Request Function Parameters	9-3
TABLE 9-2	Service Response Callback Function Parameters	9-7
TABLE 9-3	<code>send_get_request</code> Parameters	9-9
TABLE 9-4	Get Response Callback Parameters	9-15
TABLE 9-5	<code>send_set_request</code> Parameters	9-20
TABLE 9-6	Set Response Callback Parameters	9-24
TABLE 9-7	<code>send_action_req</code> Parameters	9-26
TABLE 9-8	Action Response Callback Parameters	9-31
TABLE 9-9	<code>send_create_req</code> Parameters	9-34
TABLE 9-10	Create Response Callback Parameters	9-40
TABLE 9-11	<code>send_delete_req</code> Parameters	9-44
TABLE 9-12	Delete Response Callback Parameters	9-49
TABLE 9-13	<code>send_event_req</code> Parameters	9-53

Preface

The *C++ API Reference* provides an extensive list of the classes and member methods (functions) defined in the Portable Management Interface (PMI) used to communicate with the Solstice Enterprise Manager™ (Solstice EM) Management Information Server (MIS).

Who Should Use This Book

This document is intended for programmers developing applications running with the Solstice EM MIS. A thorough working knowledge of C++ and experience using complex programmatic interfaces is assumed.

Before You Read This Book

If you have just acquired the Solstice EM product, read the *Managing Your Network* for an overview of the Solstice EM product functions, features, and components. Read the *Release Notes* for information on installing and starting the product, compatibility issues, minimum hardware and software requirements, known problems, an inventory of the product components, and late breaking information. Also refer the books listed in “Related Books section for relevant information.

How This Book Is Organized

This document is organized as follows:

Chapter 1, “Application Programming Interface,” describes the Application Programming Interfaces , Viewer, and Grapher APIs.

Chapter 2, “Common API,” describes the classes, methods, and data members that are useful when using the low- or high-level portions of the PMI.

Chapter 3, “High-Level PMI,” describes the classes and methods used for the high-level PMI.

Chapter 4, “Low-Level PMI,” describes the classes and methods used for the low-level PMI.

Chapter 5, “Access Control API,” describes the classes and methods used for the Access Control Class.

Chapter 6, “Access Control Engine API,” describes the classes and methods used for Access Control Engine Class.

Chapter 7, “Nerve Center Interface” describes the classes and methods that make up the Request Interface Library, which allows your application to build, send, and receive requests.

Chapter 8, “Topology API,” explains how to use this tool to create applications without learning the details of the MIT naming tree.

Chapter 9, “Object Services API,” explains how developers who implement object classes can use this API to describe how and when a notice is sent after receiving an event.

Related Books

Following is a list of related books:

- *Developing C++ Applications*
- *Customizing Guide*
- *Managing Your Network*

What Typographic Changes Mean

The following table describes the typographic changes used in this book.

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> You have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	<div>machine_name% su Password:</div>
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type <code>rm filename</code> .
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be root to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P-2 Shell Prompts

Shell	Prompt
C shell prompt	<code>machine_name%</code>
C shell superuser prompt	<code>machine_name#</code>
Bourne shell and Korn shell prompt	<code>\$</code>
Bourne shell and Korn shell superuser prompt	<code>#</code>

Accessing Sun Documentation Online

The `docs.sun.comsm` web site enables you to access Sun technical documentation on the Web. You can browse the `docs.sun.com` archive or search for a specific book title or subject at `http://docs.sun.com`.

Also, you can view the online documentation by pointing your browser to the following URL, `file:/opt/SUNWconn/em/docs/SEMDOCHP/index.html`

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can send your comments by email to `docfeedback@sun.com`.

Please include the part number of your document in the subject line of your email.

Application Programming Interface

This chapter describes how communication between applications can be achieved using Application Programming Interfaces (APIs).

This chapter comprises the following topics:

- Section 1.2 “Viewer API” on page 1-2
- Section 1.4 “Grapher API” on page 1-15
- Section 1.10 “Application-to-Application API” on page 1-20

1.1 API Classes

The APIs addressed in this chapter are Enterprise Manager’s Viewer, Grapher, and Application-to-Application APIs. TABLE 1-1 lists the classes described in this chapter.

TABLE 1-1 Application Programming Interface Classes

Class	API	Description
AppInstComm Class	Application-to-Application	Used to establish communication with the platform that communicates with another Solstice EM application
AppInstObj Class	Application-to-Application	Used to identify the target application for sending message
AppRequest Class	Application-to-Application	Used to request a single message
AppTarget Class	Application-to-Application	Used to send messages to applications
EMdataset Class	Grapher	Stores values or attributes

TABLE 1-1 Application Programming Interface Classes (*Continued*)

Class	API	Description
EMdynamicDataset Class	Grapher	Stores attributes of a dynamic dataset
EMgraph Class	Grapher	Creates new graphs
EMstaticDataset Class	Grapher	Stores values for graphing statically
Err Class	Grapher	Provides error checking
ViewerApi Class	Viewer	Used to communicate with applications running the Solstice EM Network Views tool

1.2 Viewer API

The Viewer API enables applications to communicate with and modify the Solstice EM Network Views tool. Platform developers can leverage both the functionality of the Network Views tool and integrate their applications with the tool. The Network Views tool, therefore, can serve as an application's central location for performing management tasks.

The Viewer API allows the application to do the following:

- Communicate with and modify the Network Views tool. For example, an application can get the current view, or set the contents of the Network Views tool's footer.
- Register with the Network Views tool to receive events generated by the tool. This involves telling the Network Views tool to send events, and, on the application's end, registering callbacks for selected events and responding to them.

The Viewer API has only one class: `ViewerApi`, in `AppInstComm`, from the Application-to-Application API.

1.3 ViewerApi Class

This section describes the member functions for the `ViewerApi` class.

1.3.1 ViewerApi Member Functions

Communication is achieved by invoking actions on application instance objects, or targets, corresponding to one or more running Network Views applications. Individual applications are represented as the `AppInstObj`, which is a member of `AppTarget` class.

`viewerapi_send_request`

```
static DU viewerapi_send_request(
    AppInstObj & obj,
    ViewerApiAction action,
    DU & action_data = DU()
)
```

This function sends a synchronous confirmed request to a specified application object and returns data. Zero is returned if there is an error. The *action* is the operation for the Network Views tool to perform and *action_data* consists of all parameters to be passed to the Network Views tool. For example, if `viewerSetCurrentZoomLevel` is the *action* and *action_data* is 20 the Network Views tool's zoom level is set to 20%.

`viewerapi_start_send_request`

```
static AppRequest viewerapi_start_send_request(
    AppInstObj & obj,
    ViewerApiAction action,
    DU & action_data = DU()
)
```

This function sends an asynchronous confirmed request to a specified *obj* and returns handles for waiting. The *action* is the operation for the Network Views tool to perform and *action_data* contains all parameters to be passed to the Network Views tool.

ViewerApi Class

viewerapi_send_request_unconfirmed

```
static Result viewerapi_send_request_unconfirmed(
    AppInstObj & obj,
    ViewerApiAction action,
    DU & action_data=DU()
)
```

This function sends an unconfirmed request to a specified *obj*, while *action* is the operation for the Network Views tool to perform and *action_data* contains all parameters to be passed to the Network Views tool.

viewerapi_build_target

```
static AppTarget viewerapi_build_target(
    Array(DU) & userid = Array(DU)()
)
```

This function builds the *target* set of viewer applications to whom API actions are sent. The *userid* is the system ID of the person running the application.

viewerapi_send_request

```
static Array (AppRequest) viewerapi_send_request(
    AppTarget & target,
    ViewerApiAction action,
    DU & action_data=DU()
)
```

This function sends a synchronous confirmed request to one or more *targets* and returns with data. Zero is returned if there is an error. The *action* is the operation for the Network Views tool to perform and *action_data* contains all parameters to be passed to the Network Views tool.

ViewerApi Class

viewerapi_start_send_request

```
static Array(AppRequest)
ViewerApi::viewerapi_start_send_request(
    AppTarget & target,
    ViewerApiAction action,
    DU & action_data=DU()
)
```

This function sends an asynchronous confirmed request to one or more *targets* and returns handles for waiting. The *action* is the operation for the Network Views tool to perform and *action_data* contains all parameters to be passed to the Network Views tool.

viewerapi_send_request_unconfirmed

```
static Result viewerapi_send_request_unconfirmed(
    AppTarget & target,
    ViewerApiAction action,
    DU & action_data=DU()
)
```

This function sends an unconfirmed request to one or more *targets*. The *action* is the operation for the Network Views tool to perform and *action_data* contains all parameters to be passed to the Network Views tool.

set_indication_handler

```
static set_indication_handler(
    AppEventHandler callback,
    const DU & pmt
)
```

This function registers a specific callback function (handler) *callback* for the event specified by *pmt*, defined as one of the GDMO PARAMETER templates shown in CODE EXAMPLE 1-1 and CODE EXAMPLE 1-2 in this chapter.

1.3.2 Communication Protocol

Two modes of communication (with an `em_viewer` application) are supported: *confirmed* and *unconfirmed*.

In the confirmed mode, the sender of the request waits for confirmation of receipt from the receiver. After confirmation is received, control is returned to the sender. The sender can also pass data to the receiver with the request, and the receiver can pass data with the response. Both synchronous and asynchronous versions are provided for confirmed requests.

In the unconfirmed mode, the request is sent to the receiving Network Views instance(s). The sender of the request neither requires nor waits for a confirmation of receipt.

Communication with an `em_viewer` application occurs in a specific order:

1. A program using `ViewerApi` sends a request to an `em_viewer`.
2. The receiving instance(s) of the Network Views tool gets the notification and performs the action.
3. The receiving instance of the Network Views tool sends a response to the sending program.
4. The sending program receives confirmation.

1.3.3 ViewerApi Actions

An application written using the `ViewerApi` class can communicate with any running viewer applications. All the actions supported by the Viewer API are described by enum `ViewerApiAction`. For any action requested, each viewer application receiver responds with a reply. TABLE 1-2 summarizes the actions.

TABLE 1-2 `ViewerApi` Class Defined Actions

Viewer API Action String	Function	Request Data Format	Reply Data Format
<code>viewerGetCurrentView</code>	Returns current view.	N/A	quoted string
<code>viewerSetCurrentView</code>	Sets Network Views tool's view.	quoted string	status (TRUE or FALSE)
<code>viewerSetFocusObject</code>	Selects object in Network Views tool.	quoted string	status
<code>viewerSetZoomLevel</code>	Sets zoom level for view.	int (%)	status

TABLE 1-2 ViewerApi Class Defined Actions (*Continued*)

Viewer API Action String	Function	Request Data Format	Reply Data Format
viewerSetFooterContents	Places a text string in the Network Views tool's footer.	quoted string	status
viewerSetViewCriteria	Sets which status variable is being used for coloring objects.	quoted string	status (TRUE or FALSE)
viewerPopupMessage	Displays a message box with the appropriate message in the Network Views tool.	quoted string	status (TRUE or FALSE)
ViewerPopupQuestion	Displays a message box with appropriate message in the Network Views tool. Sends response back to the application.	quoted string	quoted string—a comma-separated string with 1st parameter TRUE/FALSE and reset the button number clicked beginning from 0)
viewerRegisterForEvents	Registers interest in Network Views events. (See Section 1.3.4 "Event Handling" on page 1-9.)	N/A	status (TRUE or FALSE)
viewerMarkObject	Add a mark to the display of listed icons.	quoted string	status (TRUE or FALSE)
viewerUnMarkObject	Remove a mark from the display of listed icons.	quoted string	status (TRUE or FALSE)

Note – The viewerSetViewCriteria action is equivalent to selecting a status name in File→Customize→Display Settings...→Colors→User-Defined. You can also pass the argument SEVERITY to select Alarm Severity instead of User-Defined.

You can set icon colors based on values set in topoNodeDisplayStatus attribute of topology nodes. This attribute contains a list of tags and value pairs, for example, `{{"CPU", 4}, {"Diskload", 3}}`. Use the MIS Objects tool to enable tags by adding them to the topoAllStatus attribute of the object topoNodeDB=NULL. For example, `{"CPU", "Diskload"}`.

The same color mapping used for severities is used for the user-defined values, so that means the values must be between 0 and 5. You can add additional color mappings by using the MIS Objects tool to expand the range of values.

Note – The request data of the `viewerSetCurrentView` action expects a view name in the format of `system:id`. For example, `\myhost:0\`. For more information, see the *Troubleshooting Guide*.

1.3.3.1 Inputs

The `viewerPopupMessageDialog` requires the following parameters:

1. **Modality:** modal or modeless. [MODAL | MODELESS]

2. **Dialog-type:** warning, error, information or question:

[WARNING | ERROR | INFO | QUESTION]

The dialog-type action determines the look of the dialog, including the title and the icon cue used.

3. **Message-text:** The text of the message to be displayed in the dialog.

The `viewerPopupQuestionDialog` requires the same three parameters as the `viewerPopupMessageDialog`. In addition, the following parameters are required:

1. **num-buttons:** The number of buttons in the dialog.

2. **button-labels:** A sequence of *num-buttons* text labels for the buttons, from left to right.

3. **default-button:** The position of the default button to activate if the user hits the return key in the dialog. The position is a 0-based integer with 0 being the left-most button.

Note – A status of `TRUE` means that the action requested from the Network Views tool was completed successfully; a status of `FALSE` means that the action requested from the Network Views tool was not completed successfully.

1.3.4 Event Handling

Applications cannot affect only the Network Views tool; they can also listen for Network Views events, and, through callbacks, respond to those events. In this way, an application can use the Network Views tool as a central place for network management.

The general scheme is as follows:

1. The application tells the Network Views tool of the events in which it has an interest.

It does this with the `AppInstComm:set_indication_handler()` method, described in Section 1.3.1 “ViewerApi Member Functions” on page 1-3.”

For example, assume that the user has written a callback, `obj_sel_cb`, to do something when a person selects an object in the Network Views tool.

`set_indication_handler` takes the callback as its first argument, and the Network Views event as its second:

```
ViewerApi va; // create ViewerApi object
va = ViewrApi(em); // em is the Platform
                // notify Network Views tool we have a callback
                //for object selection
va.set_indication_handler(obj_sel_cb, duObjectSelectedEvent);
```

`ob_sel_cb` is called when a Network Views tool object is selected.

2. The application notifies the Network Views tool that it is interested in events.

The application sets the `ViewerAPIAction` to `viewerRegisterForEvents` (described in TABLE 1-2), then calls `send_request_unconfirmed()` (Section 1.3.1 “ViewerApi Member Functions” on page 1-3”). The *target* is the set of viewer applications to which API actions are sent.

```
// set the action to send the Network Views tool to
// 'register for events'
ViewerApiAction v_act =
va.du2ViewerApiAction("viewerRegisterForEvent");
// send the register action to the Network Views tool
va.viewerapi_send_request_unconfirmed(target, v_act, DU());
```

The syntax for `AppEventHandler::set_indication_handler()` is:

```
typedef int (IAppEventHandler)
(
    DU input_info,
    DU & reply_action,
    DU & reply_info,
    AppInstObj sender
)
```

`viewerRegisterForEvents` notifies the Network Views tool that the application is interested in events *in general*; it becomes an On switch for event notification. The Network Views tool now informs the application of *all* events, not just the ones that the application has indicated an interest in (in Step 1). Of course, the application ignores all events except the ones for which it has callbacks.

3. The Network Views tool sends events to the application.

TABLE 1-3 shows the messages that the Network Views tool sends to applications when an event occurs. See Section 1.3.5 “Network Views Messages” on page 1-11,” for more information.

TABLE 1-3 Network Views Event Messages

Network Views Message	Data Format	Event Described
<code>duObjectSelectedEvent</code>	quoted string	A user selected an object in the Network Views tool.
<code>duObjectDeselectedEvent</code>	quoted string	A user deselected an object in the Network Views tool.
<code>duPopupMenuEvent</code>	quoted string	A user brought up a Popup menu over an object.
<code>duLayerChangeEvent</code>	quoted string	The user moved a window onto another, changing the view.
<code>duToolsMenuEvent</code>	quoted string	The user selected from the Tools menu.
<code>duObjectCreationEvent</code>	quoted string	An object is added to the view being displayed.

TABLE 1-3 Network Views Event Messages *(Continued)*

Network Views Message	Data Format	Event Described
duObjectDeletionEvent	quoted string	An object is deselected from the current view.
duViewChangeEvent	quoted string	The Network Views tool changes the current view.
duRegisterForEvents	quoted string	Use this to tell the Network Views tool to send the calling application Network Views events.

4. The registered callback for that event, if any, is called. Note that an application must both:
- a. Notify the Network Views tool of specific events for which the application has callbacks, using `set_indication_handler()` (Step 1).
 - b. Notify the Network Views tool that it is interested in receiving events, using `ViewRegisterEvents` (Step 2).

TABLE 1-4 Using `ViewRegisterEvents`

Event Handlers	Interest in Receiving Event Notification Registered	Interest in Receiving Event Notification not Registered
Event Handlers Registered	Network Views tool sends all events; registered callbacks called (non-registered callbacks ignored).	Network Views tool does not send event messages.
Event Handlers Not Registered	Network Views tool notifies of all events; all events ignored.	Network Views tool does not send event messages.

1.3.5 Network Views Messages

The following describes the Network Views messages shown in TABLE 1-3. These are messages sent to applications when a Network Views event occurs.

`duObjectSelectedEvent`

This message notifies the application that a user has selected an object in the Network Views tool.

Input: This message is generated by any form of selection that the user performs in the Network Views tool (including adding an object to the current selection — that is, selecting a second object without deselecting the first). Also, when the Network Views tool changes back to a previous view that contains already selected objects, messages are sent as if the user had just selected those objects.

Processing: Whenever the Network Views tool performs a selection, it checks for any registered applications and sends them a message. If no applications have registered with the Network Views tool, no message is generated (follows the format: "system_name=unique_id").

Outputs: The action generated is `viewerObjectSelection`, and the parameter to the action is a quoted string containing the name(s) (format: 'system_name:unique_id') of the selected objects, separated by commas (for example, "name:id1,name:id2,name:id3").

`duObjectDeselectedEvent`

This action informs applications when the user deselects objects in the Network Views tool.

Inputs: Any deselection action in the Network Views tool that causes objects to be deselected generates this message. This includes any new selection action that destroys the previous selection; it also includes deleting an object that is already selected. Additionally, it includes changing views. (The Network Views tool maintains selections across views, but the application listening for selection/deselection events doesn't need to do this.)

Processing: Whenever a deselection occurs, the Network Views tool checks for any registered applications and sends them a message. If no applications have registered with the Network Views tool, no message is generated.

Outputs: The action generated is `viewerObjectDeselection`. Its parameter is a quoted string containing a comma-separated list of the "system_name:unique_ids" of the objects being deselected.

`duPopupMenuEvent`

This event allows applications to be notified when the user brings up a popup menu item over an object.

Inputs: This message results from any selection, by a user, of an item on a popup menu in the Network Views tool canvas.

Processing: Whenever a popup menu is selected, the Network Views tool determines the object on which the popup menu was “popped,” the menu item selected (the label, menu item number, and actual command associated with the popup) and generate a corresponding message. If no applications have registered with the Network Views tool, no message is generated.

Outputs: The action generated is `viewerPopupMenu`. The parameter accompanying it is a quoted string containing a comma separated list as follows:

```
'topoNodeName,menu label,menu item #,menu command'
```

`duLayerChangeEvent`

This event notifies applications when a user changes the layers currently being displayed in a view (that is, moves a window on top of or underneath another window).

Inputs: Any change of the currently selected layers (including the background image) in the current view via the `Layers Dialog` produces this message. (Note: this does *not* include the various “levels” within a geographic map.) Any view change also generates a `viewerLayerChange` message reflecting the layer flags for the new view.

Processing: When any layer change is detected, the Network Views tool generates a message that formulates a list of all the active layers *and* whether the background image is on/off and generates a message. If no applications have registered with the Network Views tool, no message is generated.

Outputs: The action generated is `viewerLayerChange`. The parameter to the message is a quoted string containing a comma-separated list of all of the layers currently being displayed. The first entry in the list always represents the background image and is either “on” or “off,” depending on the current value in the Network Views tool.

`duToolsMenuEvent`

This event notifies applications when the user has selected something from the Tools Menu.

Inputs: The user makes a selection from the Tools menu.

Processing: When the user chooses something from the *Tools Menu*, the Network Views tool generates a message based on the user’s selection. If no applications have registered with the Network Views tool, no message is generated

Outputs: The action generated is `viewerToolsMenu`. The parameter accompanying the message is a quoted string containing a comma-separated list as follows: `'menu label,menu item,menu command'`.

duObjectCreationEvent

This event notifies when any new object is added to the current view, either by the Network Views tool or from an alternative source (such as discover). These objects include links.

Inputs: This message is generated when a user creates an object within a view using

- The Network Views GUI or
- Programmatic creation of objects within the view via another PMI program (such as the Network Discovery tool).

Processing: When an object is added to the view being displayed by the Network Views tool, the Network Views tool generates a message. If no applications have registered with the Network Views tool, no message is generated.

Outputs: The action generated is `viewerObjectCreationEvent`. The parameter accompanying the message is a quoted string containing a comma-separated list as follows: `"system_name:unique_id"`.

duObjectDeletionEvent

This event message informs registered applications when an object is deleted from the Network Views tool's current view.

Inputs: This message is engendered by any object-deletion event occurring from either

- GUI manipulation within the Network Views tool, or
- Another PMI-based application (such as `em_obed`).

Processing: When an object deletion is seen by the Network Views tool, it checks to see if that object is the one currently being displayed by the Network Views tool. If it is, the Network Views tool issues a `viewerObjectDeletionMessage`.

Outputs: The action generated is a `viewerObjectDeletionEvent`. The parameter accompanying the messages is a quoted string containing a comma-separated list as follows: `"system_name:unique_id"`.

duViewChangeEvent

This event notifies applications when the user (or another application) changes the view that the Network Views tool is currently displaying.

Inputs: A change of view causes this message to be generated. Either the user of the program can change the view.

Grapher API

Processing: When the Network Views tool successfully changes the view it displays, it generates a message containing the new view name. If no applications have registered with the Network Views tool, no message is generated.

Outputs: The action generated is `viewerViewChange`. The parameter accompanying the message is a quoted string containing the “*system_name:unique_id*” of the new view being displayed.

`duRegisterForEvents`

This message is never sent by the Network Views tool to any application. See event description in TABLE 1-3.

1.3.6 Sample Programs

Sample programs using methods provided by the `ViewerApi` class are located in the `/opt/SUNWconn/em/src/viewer_api` directory. For example, the program `viewerapi_eventtester.cc` gives an example of how to use event handling with the Network Views tool.

For an example of the application instance object of a running Network Views tool, refer to `subsystemId="EM-MIS"/emApplicationID=8`. The example program, `viewerapi_driver`, uses the integer 8 to create a singular target ID. Requests can be sent to this target ID.

For more information on actions, refer to the GDMO and ASN.1 documents, `em_apps_msg.gdmo` and `em_apps_msg.asn1`.

1.4 Grapher API

A Grapher API is available with Enterprise Manager. It can be used by a client application to send data to the Graphs tool. If the Graphs tool is not running, the API will start it automatically. A grapher library (`libemgraphapi.so`) by default is located in `$EM_HOME/lib`, and the header files (`EMdataset.hh`, `EMerror.hh`, `EMgraph.hh` and `emgraphrpc.h`) are located (by default) in `$EM_HOME/include/grapher`.

A graph contains one or more datasets. There are two types of datasets: static and dynamic. The contents of a static dataset cannot be changed after it is created. A dynamic dataset is one which can be updated after it is created. A dynamic dataset can be used to plot a variable that changes with time.

Each dataset is plotted as a three dimensional bar chart by default. The X-axis represents ticks that increase in a linear fashion. Each tick on the Y-axis represents a dataset. The Z-axis represents the variable value for each dataset.

Each graph that is created has a graph name and the name of the client application associated with it. A dataset is registered only once with the Graphs tool when dynamic plotting is used. If you do not intend to use the Graphs tool for dynamic plotting, you can register the same dataset more than once. For example, you might want to do this to display graphs with the same data but with different colors or plot methods (such as absolute, cumulative, and delta) at the same time. The color of a dataset and the plot method can be changed with the Graphs tool. You can define a blank spot in a graph, called a “hole,” by specifying a large double value as the Z-value for that point. This value must be set when the dataset is created.

A sample program using the Grapher API is included with the Solstice EM product. The default directory for the sample program is `$EM_HOME/src/grapher_api`. This directory includes a README file that discusses the sample program as well as the Grapher API.

The Grapher API section includes the following classes:

- EMdataset Class
- EMdynamicDataset Class
- EMgraph Class
- EMstaticDataset Class
- Err Class

1.5 EMdataset Class

The `EMdataset` is a common base class for both `EMStaticDataset` and `EMdynamicDataset`.

1.6 EMdynamicDataset Class

An object of the class `EMdynamicDataset` represents a dataset whose variable value changes with time. A dynamic dataset can be updated with new values.

1.6.1 Constructor

```
EMdynamicDataset(
    RWCString name,           //Name of the dataset
    DynamicMode mode,         //Mode of dynamic display
```

Where *mode* is EM_DYN_ABSOLUTE, EM_DYN_CUMULATIVE, or EM_DYN_DELTA for a dataset with absolute, cumulative, or delta values.

This constructor function creates a new dynamic dataset in the Grapher API.

1.6.2 Destructor

```
~EMdynamicDataset()
```

1.7 EMstaticDataset Class

The EMstaticDataset class is for storing values to be graphed statically. The values of a static dataset cannot be updated after the dataset is created.

1.7.1 Constructor

```
EMstaticDataset(
    RWCString name, //Name of the dataset
    int numx,       //Number of X values
    double * xgrid,  //Array of X values
    double * zgrid,  //Array of Z values
    double * zvalues, //Data values for numx grid points
```

This constructor function creates a new static dataset in the Grapher.

1.7.2 Destructor

```
~EMstaticDataset()
```

1.8 EMgraph Class

The EMgraph class is for creating a new graph. The graph can consist of one or more datasets.

1.8.1 Constructor

```
EMgraph(
    char * graph_title, //Title of the graph
    char * client_name) //Name of the client creating the graph
```

```
EMgraph(
    RWCString graph_title, //Title of the graph
    RWCString client_name) //client creating the graph
```

These constructor functions create a new graph in the Grapher.

1.8.2 Destructor

```
~EMgraph()
```

1.8.3 EMgraph Member Functions

The following are member functions for the EMgraph class.

Err Class

add_dataset

```
EMhandle  
add_dataset(EMdataset * dataset) //Add a dataset
```

This function adds a dataset to the graph and returns a handle to the added dataset. The handle is necessary to add new values to the dataset.

Note – To update the dataset, it must be of the class `EMdynamicDataset`.

add_values

```
void add_values(  
    EMhandle handle,    //Handle of the dataset obtained when  
                        // dataset is added to the graph  
    double xval,        //X value  
    double zval,        //Z value
```

This function adds a new value to a dataset.

draw

```
void draw()
```

This function draws a new graph in the Grapher.

1.9 Err Class

The `Err` class provides error checking capability. Both `EMgraph` and `EMdataset` are inherited from it.

1.9.1 Member Functions

The following are member functions for the `Err` class.

GetErrType

```
ErrType GetErrType() const
```

This function returns an error type. Error types are listed in `EMerror.hh`. The sample program under `$EM_HOME/src/grapher_api` illustrates how to check error conditions.

GetErrStr

```
char* GetErrStr() const
```

This function returns an error string that can be printed or copied.

SetErrType

```
void SetErrType(ErrType)
```

This function sets an error type. The `GetErrType` function returns the set error type. Error types are listed in `EMerror.hh`. The sample program under `$EM_HOME/src/grapher_api` illustrates how to check error conditions.

1.10 Application-to-Application API

The Application-to-Application API uses the following classes to aid communication with the `emApplicationInstance` object.

- `AppInstComm` Class
- `AppInstObj` Class
- `AppRequest` Class

■ AppTarget Class

The `emApplicationInstance` object is created for any application that connects to the MIS. An application can send a message to another Solstice EM application through this object using the actions `emSendApplicationMessage` and `emSendApplicationReply`.

When an action has been sent to an application instance, the application represented by that particular `emApplicationInstance` object is notified by either an `emApplicationMessage` or `emApplicationReply` event. The emitting of the events to the applications is achieved through the behavior of the `emApplicationInstance` object.

1.11 AppInstComm Class

This class is used to establish communication with the platform that communicates with another Solstice EM application. All messages are sent and received through an instance of this class.

1.11.1 Constructors

```
AppInstComm(Platform &platform)
```

In addition to the default constructor, the above constructor is also available. It constructs an instance that maintains a connection to *platform*. The *platform* instance must be in the connected state.

1.11.2 Destructor

```
~AppInstComm( )
```

1.11.3 AppInstComm Member Functions

The following are member functions for the `AppInstComm` class.

build_target

```
static AppTarget  
build_target (  
    DU & apptype,  
    Array(DU) & userid = Array(DU)(),  
    Array(DU) & display = Array(DU)());
```

This function builds the *target* set of applications.

DataFormatter

A `DataFormatter` function can be defined for post- and pre-processing of message data. The function defined as `DataFormatter` takes the arguments *data* and *action*, does some processing, and returns new data.

The *action* is the PARAMETER defined in GDMO, while *data* is the string representation of the ASN.1 syntax defined for the PARAMETER.

```
typedef DU (*DataFormatter)(DU & data, DU & action)
```

reply_data_formatter

```
static DataFormatter reply_data_formatter
```

Allows the sender to pre-process the reply data before the sender's callback is executed.

request_data_formatter

```
static DataFormatter request_data_formatter
```

Allows the sender to pre-process the data of the message before sending it to the target application.

indication_data_formatter

```
static DataFormatter indication_data_formatter
```

Allows receiver of a message indication to pre-process the message before the receivers callback is executed.

response_data_formatter

```
static DataFormatter response_data_formatter
```

Allows the receiver of a message to pre-process the message before the reply is sent back to the caller.

send_request

```
static DU send_request(  
    AppInstObj & to,  
    DU & action,  
    DU & info=DU())
```

Sends the action with data info to the application represented by the AppInstObj class.

send_request_unconfirmed

```
static Result  
send_request_unconfirmed(  
    AppInstObj & to,  
    DU & action,  
    DU & info=DU())
```

Same as send_request but there is no reply to the message.

start_send_request

```
static AppRequest
start_send_request(
    AppInstObj & to,
    DU & action,
    DU & info=DU())
```

Same as `send_request`, but the response is asynchronous. Use the `AppRequest` member functions `begin()`, `wait()`, and `get_reply_data()` with this call.

`send_request` *for multiple targets*

```
static Array(AppRequest)
send_request(
    AppTarget & target,
    DU & action,
    DU & info=DU()
)

static Array(AppRequest)
start_send_request(
    AppTarget & target,
    DU & action,
    DU & info=DU()
)

static Result
send_request_unconfirmed(
    AppTarget & target,
    DU & action,
    DU & info=DU())
```

These member functions are the same as the previous ones, except they are used for sending requests to multiple target applications. This is achieved by using the `AppTarget` class instead of the `AppInstObj` class.

set_indication_handler

```
static AppEventHandler  
set_indication_handler(  
    AppEventHandler callback,  
    const DU & pmt  
)
```

This function calls the handler defined by *callback* when an indication of type *pmt* is received. The *pmt* parameter must be defined as one of the GDMO PARAMETER templates.

set_default_indication_handler

```
static AppAllEventsHandler  
set_default_indication_handler(  
    AppAllEventsHandler callback)
```

This function calls the handler *callback* for any application message that is not handled.

send_response

```
static Result  
send_response(  
    AppInstObj & whosent,  
    int id,  
    DU & reply_action,  
    DU & reply_data)
```

This member function is called after a successful return from an indication handler. This function is not called when the sender sends the request in unconfirmed mode.

1.12 AppInstObj Class

This class is used to identify the application to which the predefined messages are sent, and inherits properties and methods from the `Image` class. One instance of this class represents one application.

This is the class which is used to represent the target application for sending messages. If you want to send messages to multiple applications at once, use the `AppTarget` class.

1.12.1 Constructors

```
AppInstObj()
```

In addition to the above constructor, the following also applies to this class.

```
AppInstObj(int id)
```

Constructs an instance of `AppInstObj`, which represents a Solstice EM application. The argument *id* is the naming attribute of the `emApplicationInstance` object that this instance represents.

```
AppInstObj(DU & fdn)
```

Constructs an instance of `AppInstObj` which represents a Solstice EM application. The argument *fdn* is the fully distinguished name of the `emApplicationInstance` object which this instance represents.

```
AppInstObj(platform&)
```

Constructs an instance of `AppInstObj` which represents the current application.

```
AppInstObj(Image & im)
```

AppInstObj Class

Constructs an instance of `AppInstObj`, which represents a Solstice EM application. The argument *im* is an `Image` that contains the find of the `emApplicationInstance` object, which this instance represents.

1.12.2 Destructor

```
~AppInstObj()
```

1.12.3 AppInstObj Member Functions

`get_objname`

```
DU get_objname()
```

Returns the fully distinguished name of the application object instance represented by this instance of `AppInstObj`. This function is inherited from the `Image` class.

`get_oi`

```
Asn1Value get_oi()
```

Returns the fully distinguished name of the application object instance represented by this instance of `AppInstObj`. This function is inherited from the `Image` class.

Note – Other inherited functions like `get_error`, `set_error`, and `reset_error` are found in the `Image` class listings under the High-Level PMI classes. Source code references for this class and others of the `AppInst` type are found in the `app_com.hh` file.

1.13 AppRequest Class

This class represents one message. The message can be synchronous or can be asynchronous waiting to complete.

1.13.1 Constructor

```
AppRequest(AppInstObj & to, DU & action, DU & info, int id=0)
```

Destructor

```
~AppRequest()
```

1.13.2 AppRequest Member Functions

`begin`

```
Result begin()  
Result wait(Timeout api_timeout = API_DEFAULT_TIMEOUT)
```

`begin()` and `wait()` are used in conjunction with asynchronous `send_request` functions. `begin()` calls the request function and `wait()` blocks until the reply has been received for the message.

`get_reply_data`

```
DU get_reply_data()
```

After a `wait()` has returned from an asynchronous `send_request`, `get_reply_data()` is used to get the message data.

Actions

get_receiver

```
AppInstObj get_receiver()
```

Gets the `AppInstObj` that represents this `AppRequest`. This can be used when multiple replies are received.

get_action

```
DU get_action()
```

Gets the name of the message type which corresponds to this `AppRequest`. The `DU` returned contains a GDMO `PARAMETER` template name.

is_complete

```
int is_complete()
```

Returns 1 means the `AppRequest` is completed, a reply has been received. Returns 0 means the reply has not been received. This can be used in conjunction with the asynchronous `send_request` methods.

1.14 Actions

The `emSendApplicationMessage` action is used to send a message to an Solstice EM application represented by the `emApplicationInstance` object to which the action is sent.

Syntax:

```
EMSendApplicationMessage ::= SEQUENCE {  
    messageId      EMMessageID,  
    messageType    EMMessageType,  
    message        ANYDEFINEDBY messageType  
}
```

Notifications

`messageId` `EMMessageID` (`INTEGER`): Agreed-upon identifier for message.

`messageType` (`OBJECT IDENTIFIER`): This parameter is an `OID` for a `PARAMETER` type defined in GDMO. The parameter defines what the ASN.1 syntax is for the specified message type.

`emSendApplicationReply`: This action allows a Solstice EM application to send a reply to the Solstice EM application represented by this `emApplicationInstance` object.

The syntax is the same as for `emSendApplicationMessage`.

```
emSendApplicationReply ::= SEQUENCE
messageId      EMMessageID,
messageType    EMMessageType,
reply          ANYDEFINEDBY messageType
}
```

The `messageType`, `message`, and `reply` fields for these actions must be agreed upon between applications.

1.15 Notifications

The `emApplicationMessage` notification is emitted when another application performs an `emSendApplicationMessage` action on an `emApplicationInstance` object.

```
EMApplicationMessage ::= SEQUENCE {
sender                EMApplicationOI,
COMPONENTS OF        EMSendApplicationMessage
}
```

`sender` `OBJECT INSTANCE`: The `fdn` of the `emApplicationInstance` of the application which triggered the action.

COMPONENTS OF `EMSendApplicationMessage`: This contains the `messageID`, `messageType`, and message data that was part of the original action.

`emApplicationReply`: This notification is emitted when another application performs an `emSendApplicationReply` action on an `emApplicationInstance` object.

Example

Syntax:

```
EMApplicationReply ::= SEQUENCE {  
  sender          EMApplicationOI,  
  COMPONENTS OF  EMSendApplicationReply  
}
```

Both of these notifications contain the same information from the original action, and also identify the application that sent the action.

1.16 Example

Suppose you have two applications that want to keep track of each other. You have decided the best way to do this is to use an application-to-application “ping” function. (See CODE EXAMPLE 1-1 and CODE EXAMPLE 1-2 for such an application.)

You need two messages for the application: `appHello` and `appAlive`.

Following are the steps to be followed.

Example

1. Define the GDMO PARAMETER templates for these messages.

The sample below indicates how you can define these templates.

```
MODULE "EM APP PING"

appHello PARAMETER
    CONTEXT ACTION-INFO;
    WITH SYNTAX FORUM-ASN1-1.GraphicString30;
    BEHAVIOUR appHelloBehaviour;

REGISTERED AS { 1 2 3 4 5 6 90 };

appHelloBehavior BEHAVIOUR

DEFINED AS
    !Ping hello message for app to app communication!;

appReply PARAMETER
    CONTEXT ACTION-INFO;
    WITH SYNTAX FORUM-ASN1-1.GraphicString30;
    BEHAVIOUR appReplyBehaviour;

REGISTERED AS { 1 2 3 4 5 6 91 };

appReplyBehaviour BEHAVIOUR

DEFINED AS
    !Ping reply message for app to app communication!;

END
```

2. Compile the GDMO into the MIS.

```
hostname% em_gdmo hostname app_ping.gdmo
```

3. Write a driver and a listener application.

The driver sends the `appHello` message with a `GraphicString30` string as data. The listener replies with an `appReply` message containing a `GraphicString30` string as data. Make sure to link with `libappapi.so`.

4. Start the listener first.

The listener prints out its application ID. Run the driver with the ID of the listener as `arg1`.

Example

This is one way to identify what application you want to communicate with. You can also dynamically send messages to applications based on their names and what user started them. See the sample programs for more information.

CODE EXAMPLE 1-1 Listener Application

```
// Listener Application for App2App API

#include <hi.hh>
#include "app_comm.hh"

Platform plat;
int handler(DU,DU&,DU&);

main(int argc, char **argv)
{
    plat = Platform(duEM);
    char *host = "localhost";
    printf("Connecting to %s ... ",host);
    if (!plat.connect(host, "sample_app2app")) {
        printf("Connect failed\n");

        ", plat.get.error.string());
        exit(0);
    }
    printf("Connected. \n");
    AppInstComm app(plat);
    AppInstObj::appObj(plat);
    printf("ID %s\n",appObj.get_objname().chp());
    app.set_indication_handler( handler,"appHello");
    while (1)
        dispatch_recursive(TRUE);
}

int
AppEventHandler(
    DU input_info,
    DU &reply_action,
    DU &reply_info,
    AppInstObj sender
)
{
    printf("Message Received %s\n",input_info.chp());
    reply_action = "appReply";
    reply_info = "\"I am alive\"";
    return 1;
}
```

Example

CODE EXAMPLE 1-2 Driver Program for Listener Application

```
//Driver Program for App2App API

#include <hi.hh>
#include "app_comm.hh"

Platform plat;

main(int argc, char **argv)
{
    if (argc != 2) {
        printf("Usage: <listener_id>\n");
        exit(1);
    }
    plat = Platform(duEM);
    char *host = "localhost";
    printf("Connecting to %s ... ",host);
    if (!plat.connect(host, "sample_app2app")) {
        printf("Connect failed\n");
        printf("Reason:%ln", plat.get.error.string());
        exit(0);
    }
    printf("Connected. \n");
    AppInstComm app(plat);
    char    action[500];
    char    info[500];
    strcpy(action , "appHello");
    strcpy(info , "\"Hello there\"");
    AppInstObj to(atoi(argv[1]));
    DU reply;
    reply = app.send_request(to,DU(action),DU(info));
    printf("Reply is %s\n",reply.chp());
}
```

1.17 AppTarget Class

This class is used for sending messages to a set of applications based on the application name and user IDs, and inherits properties and methods from the `Album` class. An instance of this class can represent a group of applications or a single application.

1.17.1 Constructor

```
AppTarget(  
    int id,  
    DU & apptype,  
    Array(DU) & userid = Array(DU)(),  
    Array(DU) & display = Array(DU)());
```

Constructs an `AppTarget` instance that represents a set of applications based on the *apptype* and *userid*. The *apptype* is the name of the application set in the `Platform::connect()` method. The *userid* is an array of the UNIX logins of the users who connect to the platform.

1.17.2 AppTarget Member Functions

The following are member functions for the `AppTarget` class.

`num_objs`

```
int num_objs()
```

Returns the number of applications represented in the `AppTarget` instance.

AppTarget Class

first_obj

```
AppInstObj first_obj()
```

Returns the first AppInstObj in the AppTarget instance. The AppInstObj can then be used in a AppInstComm::send_request() method.

next_obj

```
AppInstObj next_obj()
```

Returns the next AppInstObj in the AppTarget instance. The AppInstObj can then be used in a AppInstComm::send_request() method.

Note – Other inherited functions like `get_error`, `set_error`, and `reset_error` are found in the `Album` class listings under the High-Level PMI classes. Source code references for this class and others of the `App` class type are found in the `app_com.hh` file.

Common API

This chapter describes classes and their member functions that can be used by all the Application Programming Interface (API) libraries.

This chapter comprises the following topics:

- Section 2.1 “Common API Classes” on page 2-1
- Section 2.2 “Class Categories” on page 2-2
- Section 2.3 “Variable Types” on page 2-3
- Section 2.4 “Class Descriptions” on page 2-4

2.1 Common API Classes

TABLE 2-1 lists the classes described in this chapter.

TABLE 2-1 Common API Classes

Class	Description
Address Class	Used to store an address
Arraydeclare Macro	Used to create a class whose structure is an array
Asn1ParsedValue Class	Represents a parsed Asn1Value that is invalidated
Asn1Tag Class	Defines an ASN.1 tag class and value
Asn1Type Class	Used to implement ASN.1 encoding and decoding
Asn1Value Class	Defines storage and operations for ASN.1 values
Blockage Class	Used to manage blocked callback events
Callback Class	Used to post and dispatch callback events
Command Class	Used to define unique commands

Class Categories

TABLE 2-1 Common API Classes (*Continued*)

Class	Description
Config Class	Used to manage database configuration file defaults
DataUnit Class	Used as a basic storage unit for data
Dictionary Class	Provides facilities to classes created by the Dictionarydeclare macro
GenInt Class	Represents integers of arbitrary size
Hash Class	
HashImpl Class	Used to implement a dynamically growing hash table
Hashdeclare Macro	Used to create a hash table class
Hdict Class	Provides facilities to classes created by the Hdictdeclare macro
Oid Class	Defines a container for an object identifier
Queue Class	
Queuedeclare Macro	Used to create a queue class with the specified type.
Timer Class	Used to schedule events to occur after a specific interval

2.2 Class Categories

There are five general categories of classes, as shown in TABLE 2-2, that can be used with the Application Programming Interface.

- Address Classes
- ASN.1 Classes
- Scheduling and Callback Handling Classes
- Array and Hashing Class
- Dictionary Macro Classes

TABLE 2-2 Class Categories

Category	Class/Macro
Address	Address Class
Address	Command Class
Address	Config Class

Variable Types

TABLE 2-2 Class Categories (*Continued*)

Category	Class/Macro
Asn1	Asn1ParsedValue Class
Asn1	Asn1Tag Class
Asn1	Asn1Type Class
Asn1	Asn1Value Class
Asn1	DataUnit Class
Asn1	Oid Class
Scheduling/Callback	Blockage Class
Scheduling/Callback	Callback Class
Scheduling/Callback	Timer Class
Arrays/Hashing	Hashdeclare Macro
Arrays/Hashing	HashImpl Class
Dictionary	Dictionary Class
Dictionary	Hdict Class

2.3 Variable Types

The basic types of variables shown in TABLE 2-3 are declared in the `/opt/SUNWconn/em/include/pmi/basic.hh` file.

TABLE 2-3 Basic Variable Types

Variable	Description
TRUE	1
FALSE	0
I8	Signed 8 bit character
U8	Unsigned 8 bit character
I16	Signed 16 Bit character
U16	Unsigned 16 Bit character
I32	Signed 32 Bit character
U32	Unsigned 32 Bit character

TABLE 2-3 Basic Variable Types (*Continued*)

Variable	Description
Boolean	General type used to distinguish true from FALSE by 0 or nonzero.
Octet	Unsigned char
Ptr	Void pointer (<code>void *</code>)
Result	Enumerated type with 2 valid values (OK, NOT_OK) used as return code in many functions.
MTime	Unsigned 32 bit quantity to hold time in Milliseconds

2.4 Class Descriptions

TABLE 2-4 lists each Common API Class with a brief description of each class.

TABLE 2-4 Common API Classes

Class	Description
Address Class	Used to store an address
Asn1ParsedValue Class	Represents a parsed Asn1Value that has not be validated against a type.
Asn1Tag Class	Defines an ASN.1 tag class and value
Asn1Type Class	Used to implement ASN.1 encoding and decoding
Asn1Value Class	Defines storage and operations for ASN.1 values
Blockage Class	Used to manage blocked callback events
Callback Class	Used to post and dispatch callback events
Command Class	Used to define unique commands
Config Class	Used to manage database configuration file defaults
DataUnit Class	Used as a basic storage unit for data
Dictionary Class	Provides facilities to classes created by the Dictionarydeclare macro
HashImpl Class	Used to implement a dynamically growing hash table

TABLE 2-4 Common API Classes (*Continued*)

Class	Description
Hdict Class	Provides facilities to classes created by the Hdictdeclare macro
Hrefdict Class	Provides facilities to classes created by the Hrefdictdeclare macro
Oid Class	Defines a container for an object identifier

2.5 Address Class

Inheritance: class Address

```
#include <pmi/address.hh>
```

This class defines a structure used to contain an address. An instance of Address contains an address class, an address tag, and an address value. The address tag is used with the value to form an address. The Address class distinguishes between members of the enum AddressClass, as shown in TABLE 2-5.

TABLE 2-5 AddressClass Data Members

Address Variables	Description
AC_DEFAULT	Route by object instance in request
AC_APP	Application address
AC_DIR_SERVICE	A directory service for resolution
AC_PRIMITIVE	Protocol driver address

The address tag can be one of the values shown in TABLE 2-6.

TABLE 2-6 AddressTag Data Members

Address Tag Variables	Value	Description
AT_PRIM_OAM	0	OAM MIS module
AT_PRIM_EMM	1	EMM MIS module
AT_PRIM_CMIP PRES_ADDR	2	CMIP address
AT_PRIM_SNMP_ADDR	3	SNMP address
AT_PRIM_AET_ADDR	4	ASN.1 Address Entity Title

TABLE 2-6 AddressTag Data Members *(Continued)*

Address Tag Variables	Value	Description
AT_PRIM_MPA_ADDR	5	MPA address
AT_PRIM_AGENT_DN	6	ASN.1 FDN
AT_PRIM_RPC_ADDR	7	RPC address
AT_PRIM_CMIP_CONFIG	8	String consisting of: {PSEL, SSEL, TSEL, NSAP}

TABLE 2-7 lists the well-known address types for directory services.

TABLE 2-7 Directory Services

Directory Service	Value	Description
AT_DS_OBJ_INST	0	Object Instance
AT_DS_APP_ENT_TITLE	1	Application Entity Title
AT_DS_DOMAIN_NAME	2	Domain

TABLE 2-8 lists the Address class public variables.

TABLE 2-8 Address Public Variables

Variable Type	Parameter	Description
AddressClass	<code>aclass</code>	The address class
AddressTag	<code>atag</code>	The address tag
DataUnit	<code>aval</code>	ASN.1-encoded address value

2.5.1 Constructor

`Address::Address()`

Default for a class is `AC_DEFAULT`. Default for a tag is `AT_PRIM_OAM`.

2.5.2 Operator

```
const Address &operator = (const Address &addr)
```

The operator above creates an instance of `Address` having the same values as the argument *addr*.

```
int operator == (const Address &a) const
```

This operator returns `TRUE` if the two `Addresses` have the same class, tag, and value.

```
int operator != (const Address &a) const
```

This operator returns `TRUE` unless the two `Addresses` have the same class, tag, and value.

2.5.3 Address Member Functions

This section describes the member functions of the `Address` class.

`print`

```
void print(FILE *fp) const  
void print(Debug &deb = misc_stdout) const
```

These function calls append a record of the `Address`'s class, tag, and value to the file *fp* or to the debug stream *deb*.

2.6 Arraydeclare Macro

```
#include <omi/array.hh>
```

```
#define Arraydeclare(T)
```

The `Arraydeclare` macro declares a class whose name is formed from “Array” followed by its argument. This permits creation of a class whose structure is an array with each element containing an instance of the class named in its argument. Because an Array object thus created is confined to the scope in which the macro is used, it deletes itself when the scope is exited. This obviates the need for “array delete” statements before return, since arrays are automatically deleted at return.

The definition for the overloaded `=` operator assures that only one Array is in charge of a piece of memory at a time. It does this by swiping the array pointer from the other Array. If you want to have multiple people referencing the same array, you must pass a reference to the Array object, or sneak the pointer out of the Array object. Be careful not to place the pointer in another Array object; that would cause a double delete. The array’s size is defined as a “pseudo const” although the class itself cheats on the `const`-ness, it expects the user not to.

Because the subscripting operator is defined in-line, there should be no extra overhead from using this class instead of subscripting directly. If you want the subscripting operator to do extra checking, define `SUBSCRIPTCHECK` appropriately.

```
#define SUBSCRIPTCHECK assert(subscript < size)
```

This macro provides validation of the subscripts used in indexing an array, but only while operating in debug mode.

The following are array declarations pre-defined in `array.hh`:

```
Arraydeclare(Boolean);
Arraydeclare(I8);
Arraydeclare(U8);
Arraydeclare(I16);
Arraydeclare(U16);
Arraydeclare(I32);
Arraydeclare(U32);
Arraydeclare(Ptr);
Arraydeclare(Result);
Arraydeclare(char);
```

2.7 Asn1ParsedValue Class

Inheritance: class Asn1ParsedValue

```
#include <pmi/asn1_type.hh>
```

Data Members: No public data members are declared in this class.

An Asn1ParsedValue represents a parsed Asn1Value that has not yet been validated against a type. This form is used to hold a parsed value until its type has been completely determined. The Asn1ParsedValue is represented internally by an Asn1Value of the type ASN-PARSED-VALUE.

Classes used in various aspects of the implementation of ASN.1 encoding and decoding rely on the ISO specifications of Abstract Syntax Notation One. TABLE 2-9 lists the Asn1ParsedValue public functions.

TABLE 2-9 Asn1ParsedValue Public Functions

Function Name	Descriptions
operator=()	
operator!()	
operator void*()	
get_real_val()	Returns the Real ASN.1 value based on the given Asn1Type.
get_parsed_val()	Returns the parsed ASN.1 value.
format_value()	Returns the parsed ASN.1 value as a string buffer.

2.7.1 Constructors

```
Asn1ParsedValue()  
  
Asn1ParsedValue(const Asn1Value &pv)  
  
Asn1ParsedValue(const Asn1ParsedValue &pv)
```

These are constructors for the Asn1ParsedValue class.

2.7.2 Asn1ParsedValue Operator Overloading

```
const Asn1ParsedValue &operator = (const Asn1ParsedValue &pv)

operator void *() const
```

These are the operators for the `Asn1ParsedValue` class.

```
int operator !() const
```

This operator is provided so that you can say “if (`!Asn1ParsedValue`)...”

2.7.3 Asn1ParsedValue Member Functions

This section describes the member functions of the `Asn1ParsedValue` class.

`format_value`

```
Result format_value(char *&buf,
    U32 &buf_len,
    U32 indent,
    Asn1Tagging tagging = TAG_EXPLICIT,
    const DataUnit def_mod = DataUnit(),
    Asn1Flags flags = 0 ) const
```

This function writes into the buffer *buf* a string representation of the `Asn1ParsedValue`. The representation must have a length no greater than *buf_len* characters, including *indent* leading blanks. This function returns OK if the value can be extracted and fits in buffer length provided.

`get_parsed_val`

```
Asn1Value get_parsed_val() const
```

This function returns the encoded parsed value as an `Asn1Value`.

Asn1Tag Class

get_real_val

```
Asn1Value get_real_val(const Asn1Type &type,  
                       Boolean indefinite = FALSE) const
```

This function, given an `Asn1Type`, returns an `Asn1Value` representation of the value.

2.8 Asn1Tag Class

Inheritance: `class Asn1Tag`

```
#include <pmi/basic.hh>  
#include <pmi/asn1_val.hh>
```

`Asn1Tag` is a class that defines an ASN.1 tag class and an ASN.1 value for a tag. An ASN.1 tag value is defined as a U32 value. The enumeration `Asn1TagClass` (see Section 2.24.7 “`Asn1TagClass`” on page 2-97”) has the following possible values:

```
CLASS_UNIV  
CLASS_APPL  
CLASS_CONT  
CLASS_PRIV
```

Following are some predefined macros which assist in making tags of specific classes. Each of the macros below creates an `Asn1Tag` instance of tag number `v`.

```
TAG_UNIV(v)  
TAG_APPL(v)  
TAG_CONT(v)  
TAG_PRIV(v)
```

Classes used in various aspects of the implementation of ASN.1 encoding and decoding rely on the ISO specifications of Abstract Syntax Notation One. For details, please consult the sources cited in the standards document.

Asn1Tag Class

TABLE 2-10 lists the Asn1Tag class public variables.

TABLE 2-10 Asn1Tag Public Variables

Variable Type	Parameter	Description
Asn1TagClass	tclass	The Asn1Tag class
Asn1TagValue	value	The value of the tag within the class

TABLE 2-11 lists the Asn1Tag public functions.

TABLE 2-11 Asn1Tag Public Functions

Function Name	Descriptions
operator==()	
operator!=()	
size()	Report the tag's size

2.8.1 Constructors

```
Asn1Tag( )
```

These are the constructors for the Asn1Tag class. The following constructor creates an instance of an Asn1Tag with a tag class of CLASS_UNIV and a tag value of 0.

```
Asn1Tag(Asn1TagClass cl,Asn1TagValue val)
```

This constructor creates an Asn1Tag with a tag class specified by *cl* and a tag value specified by *val*.

2.8.2 Asn1Tag Operator Overloading

This section describes the operators for the Asn1Tag class.

```
int operator == (const Asn1Tag &tag) const
```


Asn1Type Class

This operator compares the operand tags and returns nonzero if the two are equal, zero otherwise.

```
int operator != (const Asn1Tag &tag) const
```

This operator compares the operand tags and returns nonzero if the two are different, zero otherwise.

2.8.3 Asn1Tag Member Functions

This section describes the member function of the Asn1Tag class.

size

```
U32 size() const
```

This function call returns a U32 value containing the number of octets in the tag.

2.9 Asn1Type Class

Inheritance: class Asn1Type

```
#include <pmi/asn1_type.hh>
```

Classes used in various aspects of the implementation of ASN.1 encoding and decoding rely on the ISO specifications of Abstract Syntax Notation One. For details, please consult the sources cited in the standards document.

Associated classes: Asn1Module, ATData, Asn1TypeEL, Asn1TypeE, Asn1TypeChoice, Asn1ParsedValue, and D3SyntaxData.

TABLE 2-12 lists Asn1Type public functions.

TABLE 2-12 Asn1Type Public Functions

Function Name	Description
= void*() !()	Operator overloading
base_kind base_type kind needs_explicit primitive_type	Report kind and type
format_type	Conversion
add_tags remove_tags format_value parse_value validate validate_tag	Construct an Asn1Value from a string
cmp equivalent	Compare two values
determine_real_val	Return encoded from parsed form
find_component find_subcomponent encode	Find components
set_intersects_with set_is_subset set_add_members set_remove_members set_remove_dup_members	Set operations on Asn1Values (of the type identified by this Asn1Type)
lookup_type	Retrieve type information
register_any_handler unregister_any_handler	Provide user-defined function for the key-to-value lookup
get_range get_size_constraint get_bit_string_identifiers get_enum_identifiers	

2.9.1 Constructors

```
Asn1Type()  
Asn1Type(const Asn1Type &at)  
Asn1Type(Asn1Kind k)  
Asn1Type(const Asn1Value &av)  
Asn1Type(const DataUnit &module_name, char*, Asn1Flags flags=0 )
```

```
Asn1Type();
```

Use the preceding constructor as the default constructor.

```
Asn1Type(const Asn1Type &at);
```

Use the preceding constructor to initialize from an Asn1Value representation of the Asn1Type.

Asn1Type Class

```
Asn1Type(Asn1Kind k);
```

Use the preceding constructor to construct `BOOLEAN`, `OCTET_STRING`, `NULL`, `OBJECT_IDENTIFIER`, and `REAL` entities. `Asn1Kind` has the following numeric definitions of enumerated data types.

```
enum Asn1Kind {  
    AK_NONE, AK_BOOLEAN, AK_INTEGER, AK_BIT_STRING,  
    AK_OCTET_STRING,  
    AK_NULL, AK_SEQUENCE, AK_SEQUENCE_OF, AK_SET, AK_SET_OF,  
    AK_CHOICE, AK_SELECTION, AK_TAGGED, AK_ANY,  
    AK_OBJECT_IDENTIFIER,  
    AK_ENUMERATED, AK_REAL, AK_SUBTYPE, AK_DEFINED_TYPE  
};
```

```
Asn1Type(const Asn1Value &av);
```

Use the preceding constructor to initialize from an `Asn1Value` representation of the `Asn1Type`.

```
Asn1Type(const DataUnit &module_name, char*, Asn1Flags flags=0)
```

This constructor is initialized from the canonical text representation of the `Asn1Type`. Where *module_name* comes from the defined `DataUnit` `module_name`.

2.9.2 Destructor

```
~Asn1Type()
```

2.9.3 Asn1Type Operator Overloading

This section describes operator overloading for `Asn1Type`.

```
Asn1Type &operator = (const Asn1Type &at)
```

Asn1Type Class

This operator assigns the value of *at* to Asn1Type.

```
operator void *() const
```

This operator tests whether Asn1Type is initialized.

```
operator Asn1Value() const
```

This operator converts the Asn1Type to its Asn1Value representation.

```
int operator !() const
```

This operator is provided so that you can say “if (!Asn1ParsedValue)...”

2.9.4 Asn1Type Member Functions

This section describes the member functions of the Asn1Type class.

add_tags

```
Asn1Value add_tags(const Asn1Value &av,  
                  Boolean indefinite=FALSE) const;
```

This function call retags or adds tags to a value to make it conform to the type.

base_kind

```
Asn1Kind base_kind() const
```

This function call returns the underlying “kind” of the base type, recursing through TAGGED, SELECTION, DEFINED_TYPE, and SUBTYPE.

Asn1Type Class

base_type

```
Asn1Type base_type() const
```

This function call retrieves the underlying “type” without respect to tagging or selection.

cmp

```
int cmp(const Asn1Value &av1, const Asn1Value &av2) const;
```

This function call determines the ordering of two `Asn1Values`. Returns a negative, 0 or positive value, as *av1* is less than, equal to, or greater than *av2*. It throws an exception if an error is encountered.

determine_real_val

```
Asn1Value determine_real_val(const Asn1Value &parsed_val,  
    Boolean indefinite =FALSE, Asn1Flags flags = 0) const;
```

This function call takes (determines) an `Asn1Value` in parsed form and returns it in encoded form.

encode

```
Asn1Value encode (const Asn1Value &av,  
    Asn1Encoding encoding = ENC_BER);
```

This function call re-encodes the `Asn1Value` according to the specified encoding rules.

Asn1Type Class

equivalent

```
Boolean equivalent(const Asn1Value &av1,const Asn1Value &av2)
const;
```

This function call determines whether two Asn1Values are equivalent. It assumes that both values have been validated against the type.

find_all_components

```
Result find_all_components(const Asn1Value &val,
                           Array(DataUnit) &comp_names,
                           Array(Asn1Type) &comp_types,
                           Array(Asn1Value) &comp_vals,
                           Boolean resolve=FALSE) const;
```

This function call finds all the named components of the Asn1Type in the given Asn1Value. A component is a named field for SEQUENCE, SET, CHOICE, or a number for SEQUENCE OF, SET OF (0 is the count of the number of elements). Returns the type of the component in *comp_types*.

find_component

```
Asn1Value find_component(const DataUnit &field_name,
                         const Asn1Value &val,
                         Asn1Type &comp_type, Boolean resolve = FALSE) const;
```

This function call finds the named component of the Asn1Type in the given Asn1Value. A component is a named field for SEQUENCE, SET, CHOICE, or a number for SEQUENCE OF, SET OF (0 is the count of the number of elements). Returns the type of the component in *comp_type*.

Asn1Type Class

find_subcomponent

```
Asn1Value find_subcomponent(  
    const DataUnit &field_name,  
    const Asn1Value &val,  
    Asn1Type &comp_type, Boolean resolve = FALSE) const;
```

This function call finds the named subcomponent of the Asn1Type in the given Asn1Value. A subcomponent is a list of component names separated by a period.

format_type

```
Result format_type(char *&buf,  
    U32 &buf_len,  
    U32 indent = 0,  
    Asn1Tagging tagging = TAG_EXPLICIT,  
    const DataUnit def_mod = DataUnit(),  
    Asn1Flags flags = 0) const;
```

This function call converts the Asn1Type to its canonical text representation.

format_value

```
Result format_value(const Asn1Value &av,  
    char *&buf,  
    U32 &buf_len,  
    U32 indent = 0,  
    Asn1Tagging tagging = TAG_EXPLICIT,  
    const DataUnit def_mod = DataUnit(),  
    Asn1Flags flags = 0) const;
```

This function converts an Asn1Value to its canonical text representation.

Asn1Type Class

get_enum_identifiers()

```
Result get_enum_identifiers(Array(Asn1NamedNumber) &idents)
const;
```

This function returns a reference to an array of type `Asn1NamedNumber`, which holds the identifiers and associated values for the ASN.1 `ENUMERATED` type. If this function fails, it returns `NOT_OK`; otherwise, it returns `OK`.

get_range()

```
Result get_range(Asn1ParsedValue    &lower,
                  Boolean            &lower_open,
                  Asn1ParsedValue    &upper,
                  Boolean            &upper_open
                  ) const;
```

If this function is applied to an `Asn1Type` object that represents an ASN.1 type different from `REAL` or `INTEGER`, the function returns `NOT_OK`. If the invocation is successful, the function returns `OK`.

In addition, this function returns, by reference, the lower and upper range limits defined for a subtype of an `INTEGER` or `REAL` base type as variables of type `Asn1ParsedValue`, and the `lower_open` and `upper_open` Boolean variables that specify whether the corresponding range limit is open or closed. If the range limit is open, the `lower_open` and `upper_open` variables are set to `TRUE`; otherwise, these two variables are set to `FALSE`.

The X.208 standard defines ranges on `INTEGER` and `REAL` types. In addition, the X.208 standard uses `MIN` and `MAX` to specify lower and upper range limits respectively.

- `MIN` specifies the lower range limit defined for the parent type
- `MAX` specifies the upper range limit defined for the parent type

Asn1Type Class

The PMI library encodes MIN and MAX as NULL Asn1ParsedValue values. Therefore, after invoking the `get_range` method, you must check whether the returned values for the upper and lower range limits are NULL, before attempting to decode them; see CODE EXAMPLE 2-1.

CODE EXAMPLE 2-1 Asn1Type::get_range() Example

```
Asn1ParsedValue lower, upper;
Boolean lower_open, upper_open;
Asn1Value asnllower, asnlupper;
Asn1TypeInt int_type(AK_INTEGER);
GenInt low, up;

rslt = type.get_range(lower, lower_open, upper, upper_open);
if (rslt == OK) {
    if (lower) {
        asnllower = lower.get_real_val(int_type);
        asnllower.decode_int(low);
    } else {
        // The lower range limit is MIN
    }
    if (upper) {
        asnlupper = upper.get_real_val(int_type);
        asnlupper.decode_int(up);
    }
    } else {
        // The upper range limit is MAX
    }
}
```

kind

```
Asn1Kind kind() const
```

This function call returns the “kind” of the type, i.e, BOOLEAN, INTEGER, SEQUENCE, DEFINED_VALUE, etc.

lookup_type

```
static Asn1Type lookup_type(const DataUnit &any_name,  
                           const DataUnit &defined_by_name,  
                           const Asn1Value &any_value)
```

This function call returns the `Asn1Type` associated with a particular key value from an ANY DEFINED BY clause.

any_name is the name of the field of the sequence or set which is the ANY. *defined_by_name* is the name of the field storing the key. *any_value* is the value of the key.

An application can use this to discover the syntax of an attributes action, notification, parameter, and so on. For example, consider the type:

```
SEQUENCE {  
  key OBJECT IDENTIFIER  
  value ANY DEFINED BY KEY}
```

The following expression would return the `Asn1Type` associated with this particular combination of the three arguments.

```
Asn1Value::lookup_type("value", "key", myOID)
```

The method by which `lookup_type()` identifies a particular key-and-value combination can be provided explicitly by the static method `register_any_handler()` (below). After `register_any_handler()` has been invoked, `lookup_type()` first searches to find a function registered with an exact match in the *any_name* and *defined_by_name* fields.

Then it tries to find the function registered with a matching *any_name* but an empty *defined_by_name* field. Finally it tries to find a function with both names empty. If it can't find a function, or if the function can't locate the key, it returns a null `Asn1Type` as the result of `lookup_type()`.

Asn1Type Class

needs_explicit

```
Boolean needs_explicit() const
```

This function call determines whether the Asn1Type requires explicit tagging.

parse_value

```
Asn1Value parse_value(const DataUnit &module_name, char*,  
                      Asn1Flags = 0) const;
```

This function call constructs an Asn1Value from a given string.

register_any_handler

```
static Result register_any_handler(const DataUnit &any_name,  
                                   const DataUnit &defined_by_name, AnyHandler handler);
```

This function call specifies a key-to-value lookup function to be used by `lookup_type` (discussed above) for a particular combination of *any_name* and *defined_by_name*. Usually `register_any_handler` is invoked automatically by the PMI, but an application can invoke it explicitly.

See also: `Asn1Type::unregister_any_handler()` and `Asn1Type::lookup_type()`.

remove_tags

```
Asn1Value remove_tags(const Asn1Value &av) const
```

This function call removes explicit tags from a value to get to the underlying base value.

Asn1Type Class

set_add_members

```
Result set_add_members(const Asn1Value &of, Asn1Value &to) const;
```

This function call's arguments *of* and *to* are Asn1Values. Both of them are of the type identified by this Asn1Type. The unique members of the parameter *of* are inserted into *to*. One of the operations on SET_OF's.

Returns OK if the operation completed successfully, and NOT_OK otherwise (for example, if the Asn1Values were not both of the appropriate type).

set_intersects_with

```
Boolean set_intersects_with(const Asn1Value &left,  
                           Asn1Value &right) const;
```

This function call's arguments *left* and *right* are Asn1Values. Both of them are of the type identified by this Asn1Type. One of the operations on SET_OF's.

Returns OK if components of *left* intersect with the components of *right*; that is, if any member of *left* is a member of *right*.

set_is_subset

```
Boolean set_is_subset(const Asn1Value &left,  
                     const Asn1Value &right),  
Boolean mutual = FALSE) const;
```

This function call's arguments *left* and *right* are Asn1Values. Both of them are of the type identified by this Asn1Type. One of the operations on SET_OF's.

Returns OK if *left* is a subset of *right*; that is, if all the members of *left* are also members of *right*.

Asn1Type Class

set_remove_dup_members

```
Result set_remove_dup_members(Asn1Value &of) const;
```

This function call's argument *of* is an Asn1Values of the type identified by this Asn1Type. The method removes any duplicate members from *of*. One of the operations on SET_OF's.

Returns OK if the operation completed successfully, and NOT_OK otherwise (for example, if the Asn1Values were not both of the appropriate type).

set_remove_members

```
Result set_remove_members(const Asn1Value &of, Asn1Value &from)
const;
```

This function call's arguments *of* and *from* are Asn1Values. Both of them are of the type identified by this Asn1Type. The method removes from *from* any members that are also present in *of*. One of the operations on SET_OF's.

Returns OK if the operation completed successfully, and NOT_OK otherwise (for example, if the Asn1Values were not both of the appropriate type).

unregister_any_handler

```
static Result unregister_any_handler(const DataUnit &any_name,
    const DataUnit &defined_by_name,
    AnyHandler handler)
```

This function call undoes the effect of registering an AnyHandler function to work with lookup_type().

Asn1Value Class

validate

```
Result validate(const Asn1Value &av, Boolean ignore_tag = FALSE)
const;
```

This function call determines whether the specified `Asn1Value` is correctly formatted for the type.

validate_tag

```
Result validate_tag(const Asn1Value &av) const;
```

This function call determines whether the specified `Asn1Value` has the correct tag for the type.

2.9.5 Related Types

AnyHandler

Declared in: `asn1_type.hh`

```
typedef class Asn1Type(*AnyHandler)(const DataUnit &any_name,
const DataUnit &defined_by_name, const Asn1Value &any_value)
```

Declares a pointer to a function that looks up the `Asn1Type` of *any_value*, for a specific combination of *any_name* and *defined_by_name*.

2.10 Asn1Value Class

Inheritance: `class Asn1Value`

`#include <pmi/asn1_val.hh>`

The `Asn1Value` class defines storage and operations for ASN.1 values. The class is capable of supporting multiple encoding schemes but presently only supports Basic Encoding Rules (BER) encoding and encoding in a native machine format. The `Asn1Value` class uses the `DataUnit` class to store encoded values. It can also use other classes to store special types of data. To store unencoded data, the `Asn1Value` class makes use of the protected class `AVData` (declared as a class but not otherwise specified in `/opt/SUNWconn/em/include/pmi/asn1_val.hh`).

The `new` and `delete` operators should not be used with the `Asn1Value` constructor and destructor functions because the class manages and allocates memory for the values it stores.

2.10.1 Assignment and Data Sharing

The `=` operator is overloaded to permit assignment of one instance of `Asn1Value` to another, so that both refer to the same data.

2.10.2 Type Conversion

The `Asn1Value` class facilitates type conversion by overloading the `void *()` and `DataUnit` operators.

2.10.3 Encoding Functions

The class defines a set of encoding functions. Each allocates an `AVData` instance to store a value in native machine encoding. If the `Asn1Value` previously contained a value, the memory for it is freed prior to allocating memory for the new value. These functions normally return OK, unless memory cannot be allocated for the new value, or under certain conditions noted in the descriptions that follow.

In general, the functions do not actually perform the encoding. In most cases, the `Asn1Value` class handles the encoding automatically the first time a function requests the data in an encoded form.

2.10.4 Encoding of a Distinguished Name

A distinguished name (DN) is represented by a constructed ASN.1 value. Consider a DN containing a single RDN that has an id=1.2.6.1.2.1.42.1.13 and a value=2. It would have the following format:

T	L	V						
SEQ	x	T	L	V				
		SET	y	T	L	V		
				SEQ	z	T	L	V
						OID	a	1.2.6.1.2.1.78.1.13 (unencoded here)
						INT	b	2

A code fragment to construct the distinguished name might be as follows:

```
Octet*id = "1.3.6.1.2.1.42.1.13";

I32value = 2;

Asn1Value dn_ex1;
Asn1Value rdn_ex1;
Asn1Value ava_ex1;
Asn1Value id_ex1;
Asn1Value val_ex1;
Result status_ex1 = OK;

if (
  dn_ex1.start_construct(TAG_SEQ) != OK || // init the RDN
  rdn_ex1.start_construct(TAG_SET) != OK || // init the RDN
  ava_ex1.start_construct(TAG_SEQ) != OK || // init the AVA
  id_ex1.encode_oidstr(TAG_OID, (Octet *)id) != OK || // encode the
  OID
  val_ex1.encode_int(TAG_INT, value) != OK || // encode the integer
  ava_ex1.add_component(id_ex1) != OK || // create the AVA from
  ava_ex1.add_component(val_ex1) != OK || // ...OID and integer
  rdn_ex1.add_component(ava_ex1) != OK || // complete the RDN
  dn_ex1.add_component(rdn_ex1) != OK // complete the DN
)

status_ex1 = NOT_OK;
```

2.10.5 Decoding Simple and Constructed Asn1Values

The `Asn1Value` class defines functions for decoding both simple and constructed `Asn1Values`. If a simple `Asn1Value` stores the value in the native machine format specified by the function, it is returned directly. If the value is stored in an encoded form, it is decoded, stored in the `Asn1Value` in the decoded form (it also still exists in the encoded form) and then returns the decoded value.

The functions return `NOT_OK` if called for an uninitialized `Asn1Value`, if called for a constructed value, or if called for an `Asn1Value` that is initialized but contains neither a decoded value or encoded value of the proper type.

Classes used in various aspects of the implementation of ASN.1 encoding and decoding rely on the ISO specifications of Abstract Syntax Notation One.

TABLE 2-13 lists `Asn1Value` functions.

TABLE 2-13 `Asn1Value` Functions

Function Name	Description
<code>constructed</code>	Query <code>Asn1Value</code> info
<code>contents_size</code>	
<code>encoding</code>	
<code>incl_embedded</code>	
<code>size</code>	
<code>tag</code>	Encoding <code>Asn1Value</code> simple types
<code>encode_bits</code>	
<code>encode_boolean</code>	
<code>encode_enum</code>	
<code>encode_ext</code>	
<code>encode_int</code>	
<code>encode_null</code>	
<code>encode_octets</code>	
<code>encode_oid</code>	
<code>encode_oidstr</code>	
<code>encode_real</code>	
<code>encode_unsigned</code>	

TABLE 2-13 Asn1Value Functions (Continued)

Function Name	Description
decode_bits	Decoding Asn1Value simple types
decode_boolean	
decode_enum	
decode_ext	
decode_int	
decode_octets	
decode_oid	
decode_real	
decode_unsigned	
add_component	Encoding constructed Asn1Values
make_explicit_tagged	
start_construct	
delete_component	Decoding constructed Asn1Values
first_component	
make_explicit_tagged	
next_component	
retag	
tagged_component	

2.10.6 Constructors

```
Asn1Value();
```

The following constructor allocates storage for an uninitialized ASN.1 value. Memory is not allocated for an AVData instance or for a DataUnit instance. The constructor creates an instance of an Asn1Value and sets its AVData pointer to 0.

No memory is allocated for the AVData instance or for a DataUnit in the constructor. If the encoding specified by the *enc* parameter matches the encoding specified for Asn1Value specified by the *av* parameter, the constructor instantiates an Asn1Value that points to the AVData contained in *av* (that is, it shares data with *av*). The reference count for the AVData instance pointed to by *av* is incremented. The default encoding for this constructor follows the Basic Encoding Rules (BER); currently only BER is supported.

```
Asn1Value(const Asn1Value &av);
```

The new operator should not be used with the above constructor function.

Asn1Value Class

In the following constructor, memory is allocated for an AVData instance and the AVData instance is pointed at the DataUnit referenced by *du*. The reference count for the AVData instance created is set to one. No memory is allocated for the DataUnit. If the DataUnit specified by *du* is not valid, the pointer to the AVData instance is set to 0 and the memory for the AVData instance is deallocated.

```
Asn1Value(const DataUnit &du);
```

The new operator should not be used with the above constructor function.

In the following constructor, memory is allocated for an AVData instance, but the DataUnit is shared with the DataUnit specified by the *du* parameter. The tag for the Asn1Value is set to the value specified by the *tag* parameter.

An Asn1Value can be a constructed type or a simple type. The *constr* parameter specifies whether the value specified by *du* is a constructed type. The *type* attribute of the newly instantiated Asn1Value is set to the value specified by *constr*.

```
Asn1Value(const Asn1Tag &tag,  
          Boolean constr,  
          const DataUnit &du);
```

The new operator should not be used with the above constructor function.

2.10.7 Destructor

```
~Asn1Value();
```

The destructor decrements the reference count for the AVData instance pointed at by this Asn1Value. If the reference count goes to zero, the memory for the AVData instance is deallocated.

2.10.8 Asn1Value Operator Overloading

The `=` operator is overloaded so that one `Asn1Value` can share data with another `Asn1Value`. In the returned `Asn1Value`, the `AVData` pointer points to the same object as the `AVData` pointer in `av`. At the same time, assignment increases the reference count for the `AVData` unit that `av` points to.

```
Asn1Value &operator = (const Asn1Value &av);
```

If the instance of the `Asn1Value` that this function is operating was pointed to another `AVData` instance, the reference count for the instance is decremented; if the reference count goes to zero, the memory for instance is deallocated. The `'='` operator returns a reference to an `Asn1Value`.

```
operator void *() const;
```

The operator `void *` returns a void pointer to the `AVData` instance pointed to by the `Asn1Value`. This function performs a type conversion from an `Asn1Value` to a `void *` and can be used for implicit type conversions.

```
operator DataUnit() const;
```

The operator `DataUnit` performs a type conversion from an `Asn1Value` to a `DataUnit`. If the `Asn1Value` is uninitialized, it returns a zero length `DataUnit`. This function can be used for implicit type conversions.

```
int operator !() const
```

This allows you to say `"if(!Asn1Value...)"`

2.10.9 Asn1Value Member Functions

This section describes the member functions of the `Asn1Value` class.

Asn1Value Class

add_component

```
Result add_component(Asn1Value &comp)
```

This function call adds a component to this constructed Asn1Value. Alternatively,

```
Result add_components(const class Array(Asn1Value) &comps)
```

returns OK if the operation completes normally, but NOT_OK if it encounters any of the following conditions:

- The Asn1Value has not been initialized;
- The Asn1Value is not a constructed type;
- The Asn1Value specified by component has not been initialized;
- The encoding of Asn1Value specified by component does not match the encoding of the Asn1Value to which the component is being added; or
- Memory cannot be allocated to add the new component.

The size of the constructed value is updated to reflect the size of the component(s) added.

compute_total_size

```
void compute_total_size(U32 &size);
```

This function call reports on Memory size information.

constructed

```
Boolean constructed() const
```

This function call returns a boolean value indicating if the Asn1Value is a constructed type. (An Asn1Value can either be a constructed type or a simple type.)

Asn1Value Class

contents_size

```
Result contents_size(U32 &sz, Boolean inc_embed = FALSE) const
```

This function call sets the value of the *sz* parameter to the total size of a constructed *Asn1Value*. This function decodes each component of a constructed *Asn1Value* and returns the sum of the size of each *DataUnit* in the constructed value. If the *Asn1Value* does not decode properly, this function returns *NOT_OK*; otherwise it returns *OK*.

decode_bits

```
Result decode_bits(DataUnit &du, U32 &len) const
```

This function call decodes an encoded boolean value and stores the result in the *DataUnit* specified by the *du* parameter. The length of the bitstring, in bits, is assigned to the parameter specified by *len*.

```
Result decode_bits(Octet *val, U32 &len) const
```

This function call decodes an encoded boolean value and store the result in the Octet string pointed to by the *val* parameter. The length of the bit string, in bits, is assigned to the parameter specified by *len*.

decode_boolean

```
Result decode_boolean(Boolean &val) const
```

This function call decodes an encoded boolean value and stores the result in the variable referenced by *val*.

decode_enum

```
Result decode_enum(I32 &val) const;
```

This function call decodes an enumerated value and stores the result in the I32 variable referenced by the *val* parameter.

```
Result decode_enum(Asn1Int &val) const;
```

Alternatively, this function call decodes an enumerated value and stores the result in the Asn1Int variable referenced by the *val* parameter.

decode_ext

```
Result decode_ext(Oid &oid,
                  I32 &indirect,
                  DataUnit &odes,
                  Asn1Value &encoding) const
```

This function call decodes an ASN1 external value, decomposing it into its components by setting the variables shown in TABLE 2-14 below.

TABLE 2-14 decode_ext Variable Descriptions

Variable	Description
oid	Its (optional) object identifier
indirect	Its (optional) indirect reference
odes	Its (optional) data-value descriptor
encoding	Its choice of the following, indicated by its tag: [0] ANY [1] Implicit octet string [2] Implicit bit string

Asn1Value Class

decode_int

```
Result decode_int(I32 &val) const;
```

This function call decodes an encoded I32 value and stores the result in the variable referenced by *val*.

```
Result decode_int(Asn1Int &val) const;
```

Alternatively, this function call decodes an encoded Asn1Int value and stores the result in the variable referenced by *val*.

decode_octets

```
Result decode_octets(DataUnit &du) const;
```

This function decodes an encoded octet string.

```
Result decode_octets(Octet *val, U32 &len) const;
```

This function call decodes an octet string of length *len*, and stores a copy of the decoded string into the Octet string pointed to by *val*.

decode_oid

```
Result decode_oid(Oid &oid) const;
```

These function calls decode an OID.

```
Result decode_oid(Octet *val, U32 &len) const;
```

These function calls decode an OID (Object IDentifier). The argument can be the address of an OID, or a pointer to an octet string and the string's length. The decoding is formed by calling `decode_octets`. If the `oid` is not constructed, the function returns `NOT_OK`; otherwise it returns the result of `decode_octets`.

decode_real

```
Result decode_real(double &val) const;
```

These function calls decode an encoded real value and stores the result in the variable referenced by *val*, which takes the form of the specified type. The double integer is specified above.

For I32 scientific notation,

```
Result decode_real(I32 &mantissa, U8 &base, I32 &exponent) const;
```

For Asn1Int scientific notation,

```
Result decode_real(Asn1Int &mantissa, U8 &base,  
                  Asn1Int &exponent) const;
```

decode_unsigned

```
Result decode_unsigned(U32 &val) const;
```

This function call decodes an encoded U32 value and stores the result in the variable referenced by *val*.

delete_component

```
Result delete_component(const Asn1Value &zap, Asn1Value &next)
```

This function call finds the component specified by *zap*, and removes it from the *Asn1Value*. The routine assigns the component that originally followed the deleted component to the *Asn1Value* specified by *next*. If no components remain following the one specified by *zap*, *next* is assigned an uninitialized *Asn1Value*, but the function still returns OK. Internally, this function performs lazy decoding, and only decodes values into components as they are needed.

Returns NOT_OK if the *Asn1Value* is not initialized, or if the *Asn1Value* can not be decoded into separate components.

Asn1Value Class

drop_encoding

```
void drop_encoding();
```

This function call will remove any cached values that have been encoded.

encode_bits

```
Result encode_bits(const Asn1Tag &tag, const Octet *val, U32 len);
```

```
Result encode_bits(const Asn1Tag &tag,  
                  const DataUnit &val, U32 len);
```

These function calls copy and store the value specified by *val* into the *Asn1Value*. The length of *val*, in bits, is specified by *len*. The bit value specified by *val* is expected to be passed to this function as an ASN.1 bit value.

If the *Asn1Value* previously contained a value, the storage for the previous value is deallocated before the new value is stored. The tag for the *Asn1Value* is set to the value specified by the *tag* parameter. The encoding of the *Asn1Value* defaults to BER.

Returns `NOT_OK` if memory cannot be allocated to store the encoded bit value.

encode_boolean

```
Result encode_boolean(const Asn1Tag &tag, const Boolean val);
```

This function call stores a copy of the boolean value specified by the *val* parameter in the *Asn1Value*. If the *Asn1Value* previously contained a value, the storage for the previous value is deallocated before the new value is stored. The tag for the *Asn1Value* is set to the value specified by the *tag* parameter. The encoding of the *Asn1Value* is assumed to be BER.

encode_enum

```
Result encode_enum(const Asn1Tag &tag, const I32 val);
```

```
Result encode_enum(const Asn1Tag &tag, const Asn1Int &val);
```

These function calls store a copy of the value specified by *val* in the Asn1Value. If the Asn1Value previously contained a value, the storage for the previous value is deallocated before the new value is stored. The tag for the Asn1Value is shared which is set to the tag value specified by the tag parameter. The encoding of the Asn1Value defaults to BER.

Returns NOT_OK if storage cannot be allocated for the Asn1Value.

encode_ext

```
Result encode_ext(const Asn1Tag &tag,
    const Oid &oid
    I32 indirect,
    const DataUnit &odes,
    Asn1Value &encoding,
    Boolean indefinite = FALSE);
```

This function call inserts into this instance of Asn1Value an encoding of the EXTERNAL type, as defined by ISO 8824/X.208.

In that standard, the EXTERNAL type is defined as follows:

```
[UNIVERSAL 8] IMPLICIT SEQUENCE
{
    direct-reference      OBJECT IDENTIFIER OPTIONAL,
    indirect-reference    INTEGER OPTIONAL,
    direct-reference      ObjectDescriptor OPTIONAL,
    encoding              CHOICE
        {
            single-ASN1-type [0] ANY
            octet-aligned     [1] IMPLICIT OCTET STRING,
            arbitrary         [2] IMPLICIT BIT STRING
        }
}
```

The arguments that are not applicable to a particular case can be empty, provided that not all of *tag*, *oid*, *indirect*, *odes*, and *encoding* are empty.

TABLE 2-15 describes the `encode_ext` arguments:

TABLE 2-15 `encode_ext` Arguments

<i>tag</i>	Tag identifying the class of encoding
<i>oid</i>	Object identifier
<i>indirect</i>	Indirect reference
<i>odes</i>	Object descriptor
<i>encoding</i>	ASN1 encoding
<i>indefinite</i>	Whether this is an indefinite-length encoding

Returns `OK` if a value based on the arguments is successfully inserted into this ASN1 value.

`encode_int`

```
Result encode_int(const Asn1Tag &tag, const I32 val);
```

```
Result encode_int(const Asn1Tag &tag, const Asn1Int &val);
```

These function calls store a copy of the value specified by the *val* parameter in the `Asn1Value`. If the `Asn1Value` previously contained a value, the storage for the previous value is deallocated before the new value is stored. The tag for the `Asn1Value` is shared with is set to the tag value specified by the *tag* parameter. The encoding of the `Asn1Value` is set to the encoding specified by the *enc* parameter, but defaults to BER if not specified.

`encode_minus_infinity`

```
Result encode_minus_infinity(const Asn1Tag &tag);
```

This function call encodes an `Asn1Value` having a tag specified by *tag* which may be a large negative number in the set of possible `Asn1` numbers, larger than permitted by the limits of the compiler's integer type.

encode_null

```
Result encode_null(const Asn1Tag &tag);
```

This function call encodes a zero-length Asn1Value having a tag as specified by *tag*. The encoding of the Asn1Value defaults to BER. If the Asn1Value previously contained a value, the storage for the previous value is deallocated before the null value is stored. This function allocates storage for an AVData instance, but does not allocate storage for a DataUnit.

Returns NOT_OK if storage cannot be allocated for the AVData instance.

encode_octets

```
Result encode_octets(const Asn1Tag &tag,  
                    const Octet *val, U32 len);
```

```
Result encode_octets(const Asn1Tag &tag,  
                    const DataUnit &val);
```

These function calls store a copy of the value specified by *val* in the Asn1Value. The length of the octet value, in octets, is specified by *len*. DataUnit has no specified length. If the Asn1Value previously contained a value, the storage for the previous value is deallocated before the new value is stored. The tag for the Asn1Value is set to the tag value specified by *tag*. The encoding of the Asn1Value defaults to BER. Returns NOT_OK if memory cannot be allocated to store the octet value.

The alternate form of the function call shares data with the DataUnit specified by *val*. The tag for the Asn1Value is set to the tag value specified by *tag*. The encoding of the Asn1Value defaults to BER. If the Asn1Value previously contained a value, the storage for the previous value is deallocated before the new value is stored.

Returns NOT_OK if memory cannot be allocated to store the portion of the Asn1Value that points to the shared DataUnit.

Asn1Value Class

encode_oid

```
Result encode_oid(const Asn1Tag &tag,  
                 const Octet *val,  
                 U32 len);
```

```
Result encode_oid(const Asn1Tag &tag,  
                 const Oid &val);
```

These function calls encode and store an OID as an ASN.1 `oid`. The OID can be either the address of an OID, or an octet string and length. The tag of the encoded `oid` is set to the tag specified by *tag*.

The encoding of the `Asn1Value` is BER (`ENC_BER`), which is also the default encoding. If any other encoding is specified, the method returns `NOT_OK`. It also returns `NOT_OK` if it cannot allocate the temporary storage needed as part of the encoding process.

encode_oidstr

```
Result encode_oidstr(const Octet *dot_str,  
                    const Asn1Tag &tag=TAG_OID);
```

This function call encodes and stores an integer dot string (for example, “1.3.6.1.2.1.78”) as an ASN.1 `oid`. The tag of the encoded `oid` is set to the tag specified by *tag*. The encoding of the `Asn1Value` is BER (`ENC_BER`), which is also the default encoding. Any other encoding causes the function to return `NOT_OK`. The function also returns `NOT_OK` if it cannot allocate the temporary storage needed as part of the encoding process.

encode_plus_infinity

```
Result encode_plus_infinity(const Asn1Tag &tag);
```

This function call encodes an `Asn1Value` having a tag specified by *tag* which may be a large positive number in the set of possible Asn1 numbers, larger than permitted by the limits of the compiler’s integer type.

encode_real

```
Result encode_real(const Asn1Tag &tag,  
                  double val);
```

```
Result encode_real(const Asn1Tag &tag, I32 mantissa,  
                  U8 base, I32 exponent);
```

```
Result encode_real(const Asn1Tag &tag, const Asn1Int &mantissa,  
                  U8 base, const Asn1Int &exponent);
```

The first function stores a copy of the value specified *val* into the *Asn1Value*. The second and third functions store the value derived from the *mantissa*, *base*, and *exponent* parameters. The tag for the *Asn1Value* is set to the value specified by *tag*. The *Asn1Value*'s encoding defaults to BER. If the *Asn1Value* previously contained a value, the storage for the previous value is deallocated before the new value is stored.

Returns NOT_OK if storage cannot be allocated for the *Asn1Value*.

encode_unsigned

```
Result encode_unsigned(const Asn1Tag &tag, const U32 val);
```

This function call stores a copy of the unsigned integer value specified by the *val* parameter in the *Asn1Value*. If the *Asn1Value* previously contained a value, the storage for the previous value is deallocated before the new value is stored. The tag for the *Asn1Value* is set to the tag value specified by the *tag* parameter. The encoding of the *Asn1Value* defaults to BER.

first_component

```
Result first_component(Asn1Value &comp) const
```

This function call assigns the first component in an *Asn1Value* to the *Asn1Value* specified by *comp*. The function returns NOT_OK if the *Asn1Value* is not initialized, or if the *Asn1Value* can not be decoded into separate components.

Asn1Value Class

get_components

```
Result get_components(class Array(Asn1Value) &comps) const;
```

This function call finds all the components in an `Asn1Value` and assigns them to the `Asn1Value` array specified by *comps*.

The function returns `NOT_OK` if the `Asn1Value` is not initialized, or if the `Asn1Value` can not be decoded into separate components.

indefinite_length

```
Boolean indefinite_length() const;
```

This function call returns `FALSE` if the `Asn1Value` is not initialized, or if the `Asn1Value` can not be decoded into separate components or components of a specified length.

make_explicit_tagged

```
Result make_explicit_tagged(const Asn1Tag &tag,  
    const Asn1Value &comp,  
    Boolean indefinite = FALSE);
```

This function call re-constructs the `Asn1Value` with the input *comp* `Asn1Value`, and retags it with *tag*.

The function returns `NOT_OK` if the `Asn1Value` is not initialized, or if the `Asn1Value` can not be decoded into separate components.

Asn1Value Class

next_component

```
Result next_component(const Asn1Value &prev, Asn1Value &next)
const
```

This function call finds the component specified by *prev*, and assigns the component that follows it to the *Asn1Value* specified by the *next* parameter. If no components remain following the one specified by *prev*, *next* is assigned an uninitialized *Asn1Value*, but the function still returns OK. If no components remain, an empty *Asn1Value* is returned.

Internally, this function performs lazy decoding, and only decodes values into components as the components are needed. Returns NOT_OK if the *Asn1Value* is not initialized, or if the *Asn1Value* can not be decoded into separate components.

num_comps

```
U32 num_comps() const;
```

This function call returns the number of components of the *Asn1Value*.

print

```
void print(FILE *fp, U32 indent = 0) const;
```

```
void print(Debug &deb = misc_stdout, U32 indent = 0) const;
```

```
void print(char *c, U32 indent = 0) const;
```

Each of the preceding function calls prints a formatted record of the *Asn1Value* class, appending it either to the file whose pointer is *fp*, or the debug stream *deb*, or to the character string pointed to by *c*.

Asn1Value Class

retag

```
Result retag(const Asn1Tag &tag)
```

This function call replaces the `Asn1Value`'s previous tag with the *tag* specified in the argument.

size

```
U32 size()const;
```

This function call returns a `U32` value containing the length of an encoded `Asn1Value`. If the `Asn1Value` is uninitialized, it returns a length of zero. The size returned is the length `DataUnit` in which the encoded value is stored

start_construct

```
Result start_construct(const Asn1Tag &tag,  
                      Boolean indefinite = FALSE)
```

This function call initiates the creation of a constructed `Asn1Value`. The tag of the constructed type is set to the value specified by *tag*. If the `Asn1Value` previously contained a value, the storage for the previous value is deallocated before the new value is stored. The constructor sets a flag value to indicate whether the `Asn1Value` contains a constructed type; `start_construct` also sets the size of the value to zero.

Tag

```
const Tag &tag()const;
```

This function call returns a reference to the `Asn1Value`'s tag.

tagged_component

```
Result tagged_component(Asn1Value &comp) const;
```

This function call checks to see if an `Asn1Value` contains only a single component. If the `Asn1Value` contains a single, initialized component, it assigns the component to the `Asn1Value` specified by the *comp* parameter.

If the `Asn1Value` contains more than one component, or is uninitialized, the function returns `NOT_OK`.

2.10.10 Related Global Functions

enc

```
extern Asn1Value enc(  
    char *module,  
    char *type,  
    char *data  
);
```

The `enc` function returns an Asn1 encoded value of *data* based on the *module* and *type* information, or returns a NULL `Asn1Value` in case of an error.

getGeneralizedTime

```
extern Result getGeneralizedTime(  
    Asn1Value &gt,  
    const Asn1Tag &tag  
);
```

The `getGeneralizedTime` function gets the local time as a generalized time. The *gt* parameter indicates where to store the generalized time, and *tag* specifies the tag to be used. The function returns `OK` when the generalized time is stored where specified in *gt*, or `NOT_OK` if there is an error.

2.11 Blockage Class

Inheritance: `class Blockage`

```
sched.hh
```

An instance of the `Blockage` class is a queue of callback events that are currently blocked and must be handled when they become unblocked. The class provides methods by which you can instruct the `Blockage` function on how to deal with the callbacks on its list as they become unblocked, or force their invocation at once.

The scheduler uses an instance of `Blockage` to maintain the scheduler queue, called `sched_q`, which is declared as `extern`.

`sched_q`

Declared in: `sched.hh`

```
extern Blockage sched_q;
```

The functions `post_callback()`, `purge_callback()`, `purge_callback_handler()`, and `purge_callback_data()` use `sched_q`, the `extern` instance of `Blockage` used to manage scheduler events.

Events in the scheduler queue are blocked only until the next time the scheduler runs. The scheduler dispatches callbacks before calling `select()` and before returning. Callbacks are invoked in the order in which they were posted (FIFO).

TABLE 2-16 Blockage Public Functions

Post to a wait queue	One new callback	<code>sleep</code>
Invoke	All waiting callbacks	<code>wakeup</code> <code>wakeup_now</code>
Invoke	Any matching callbacks	<code>wakeup</code>
Invoke	Any callbacks with matching handler	<code>wakeup</code>

TABLE 2-16 Blockage Public Functions (*Continued*)

Invoke	Any callbacks with matching data	wakeup
Invoke or purge	Any callbacks with matching call data	wakeup_call purge_call
Retrieve	Number of callbacks in the queue	size

2.11.1 Constructor

`Blockage()`

Establishes a new instance with an empty queue of callback events.

2.11.2 Blockage Member Functions

`purge_call`

`void purge_call(Ptr cdata)`

Purges from the Blockage's queue those callbacks whose callback data matches *cdata*.

`size`

`U32 size()`

Returns the number of members in the blockage's queue.

`sleep`

`void sleep(const Callback &e, Ptr call_data=0)`

Posts a callback to the Blockage's wait queue.

Blockage Class

wakeup

```
void wakeup()
```

This function invokes all of the Blockage's waiting callbacks.

```
void wakeup(Callback &e)
```

Invokes the callback event identified by *e*.

```
void wakeup(CallbackHandler handler)
```

Invokes any of the Blockage's callbacks whose handler matches *handler*.

```
void wakeup(Ptr data)
```

Invokes any of the Blockage's callbacks whose data matches *data*.

wakeup_call

```
void wakeup_call(Ptr cdata)
```

Invokes any of the Blockage's callbacks whose callback data matches *cdata*.

wakeup_now

```
void wakeup_now()
```

Invokes immediately all waiting callback events in the Blockage's queue.

2.11.3 Related Global Functions

post_callback

```
inline void post_callback(const Callback &e, Ptr cdata=0_)
```

Adds a `Callback` instance to the end of the scheduler's immediate callback queue, optionally with callback data *cdata*. The scheduler dispatches callbacks just before calling `select()` and just before returning. Callbacks will occur in the order posted.

purge_callback

```
inline void purge_callback(const Callback &e)
```

This function purges any matching callbacks from the scheduler queue. The scheduler will not dispatch these callbacks the next time it is run.

purge_callback_data

```
inline void purge_callback_data(Ptr data)
```

This function removes all callbacks whose `USER` data matches the parameter *data* from the scheduler callback queue. It calls the (overloaded) operator `==` to do the comparison.

purge_callback_cdata

```
inline void purge_callback_cdata(Ptr data)
```

This function removes all callbacks whose `CALL` data matches the parameter *data* from the scheduler callback queue.

Blockage Class

`purge_callback_handler`

```
inline void purge_callback_handler(CallbackHandler handler)
```

This function purges all callbacks with matching handler from the scheduler queue.

`post_fd_read_callback`

```
void post_fd_read_callback(int fd, const Callback &cb);
```

The `post_fd_read_callback` function posts callback *cb* to the scheduler's read callback queue for file descriptor *fd*. When the scheduler detects (by `select()`) something ready to read on *fd*, the callback *cb* gets called.

`post_fd_write_callback`

```
void post_fd_write_callback(int fd, const Callback &cb);
```

The `post_fd_write_callback` function posts callback *cb* to the scheduler's write callback queue for file descriptor *fd*. When the scheduler detects (by `select()`) something ready to write on *fd*, the callback *cb* gets called.

`post_fd_except_callback`

```
void post_fd_except_callback(int fd, const Callback &cb);
```

The `post_fd_except_callback` function posts callback *cb* to the scheduler's exception callback queue for file descriptor *fd*. When the scheduler detects (by `select()`) any exception on *fd*, the callback *cb* gets called.

Blockage Class

`purge_fd_read_callback`

```
void purge_fd_read_callback(int fd);
```

The `purge_fd_read_callback` function calls `FD_CLR` on *fd* for the read mask. Any previously posted read callback will not get called by the scheduler.

`purge_fd_write_callback`

```
void purge_fd_write_callback(int fd);
```

The `purge_fd_write_callback` function calls `FD_CLR` on *fd* for the write mask. Any previously posted write callback will not get called by the scheduler.

`purge_fd_except_callback`

```
void purge_fd_except_callback(int fd);
```

The `purge_fd_except_callback` function calls `FD_CLR` on *fd* for the exception mask. Any previously posted exception callback will not get called by the scheduler.

`purge_fd_callbacks`

```
void purge_fd_callbacks(int fd);
```

The `purge_fd_callbacks` function calls `FD_CLR` on *fd* for the read, write, and exception masks. Any previously posted read, write, or exception callbacks will not get called by the scheduler.

`flush_events_callbacks`

```
void flush_events_callbacks();
```

The `flush_events_callbacks` function allows applications to flush all events to the MIS by executing all callbacks in both scheduler read and write callback queues.

2.12 Callback Class

Inheritance: `class Callback`

Declared in: `callback.hh`

This class serves to post and dispatch callback events. Callbacks are declared by the application and posted to the scheduler by the various posting routines. A callback contains a function pointer specifying the handler to be called when the callback is eventually dispatched. When a callback handler is called, it is passed two pieces of data:

- `user_data`: The data placed in the Callback by the application
- `call_data`: Supplied by the routine dispatching the call (and can be NULL)

The scheduler keeps a queue of callbacks that should happen immediately. The scheduler routine dispatches these callbacks once the routine is entered, and again after it has dispatched all I/O and timeouts, in case the I/O or timeout callbacks scheduled any immediate callbacks themselves. Immediate callbacks occur in the order posted.

TABLE 2-17 Callback Public Variables

<code>CallbackHandler handler;</code>	The function the scheduler should call
<code>Ptr data;</code>	Data provided as argument to the callback function

TABLE 2-18 Callback Public Functions

Operator overloading	<code>void*</code>
Invoke the callback directly	<code>exec</code>

The callback class has the following typedef:

CallbackHandler

```
typedef void (*CallbackHandler)(Ptr user_data, Ptr call_data)
```

2.12.1 Constructor

```
Callback()
```

Creates a callback instance with no handler or data.

```
Callback(CallbackHandler hand, Ptr d)
```

Creates a callback instance with handler *hand* and data *d*.

2.12.2 Callback Operator Overloading

```
friend int operator == (const Callback &e1, const Callback &e2)
```

Returns `TRUE` if the two callbacks have equal data and the same context and the same callback handler; `FALSE` otherwise.

```
operator void*() const
```

Returns a pointer to the callback's handler.

2.12.3 Callback Member Functions

`exec`

```
void exec(Ptr call_data)
```

This function invokes the callback's handler with the callback's data and `exec`'s *call_data* as arguments. The callback handler is executed, in its own Try block, with `CATCHALL` set to write a record of any exceptions to `misc_stderr.print`.

2.13 Command Class

Inheritance: `class Command`

`#include <pmi/command.hh>`

Data Members: No public data members are declared in this class.

The `Command` class is used to define unique commands that can be passed between entities. The address of the `Command` object itself is used to identify it uniquely from all other `Command` objects. A `Command` object should only be defined at the global scope, and should always be `const`.

2.13.1 Constructor

```
Command();
```

This is the constructor for the `Command` class.

2.13.2 Operator

Following is the operator for the `Command` class. It casts a `Command` instance to an unsigned integer.

```
operator U32() const;
```

2.14 Config Class

Inheritance: `class Config`

`#include <pmi/config.hh>`

Data Members: No public data members are declared in this class.

The `Config` class is used to define the orderly management of per-program or per-MIS defaults from configuration database files.

Config Class

Each instance of the `Config` class has its own mapping of default names to default values. This mapping is loaded from a file defined either in the constructor or as a result of a call to the `load` member function.

Each line in the file is of the form:

```
key : value
```

Initial and trailing whitespace are ignored. A comment sign ('#') can be used to indicate that the remainder of the line should be ignored. TABLE 2-19 lists the `Config` public functions.

TABLE 2-19 Config Public Functions

Function Name	Descriptions
<code>fetch</code>	Retrieve a default data value
<code>load</code>	Load a configuration database

2.14.1 Constructors

```
Config()  
  
Config(const char *file)
```

These are both constructors for the `Config` class.

2.14.2 Config Member Functions

This section describes the member functions of the `Config` class.

DataUnit Class

load

```
void load(const char *env, const char *file)
```

This function call loads the configuration file into the mapping table. If the environment variable *env* has been set, the function loads the file corresponding to *env*'s value. If it is not set, the file `$EM_MIS_HOME/file` is used. If `$EM_MIS_HOME` is not set, the local directory copy of *file* is used.

```
void load(const char *file)
```

This function call loads the configuration file *file* into the mapping table.

fetch

```
const char *fetch(const char *key, const char *dflt = 0)
```

This function call looks up a mapping in the table. If the mapping does not exist, the value *dflt* is returned. Otherwise, returns a pointer to the string representing the value.

2.15 DataUnit Class

```
#include <pmi/du.hh>
```

Inheritance: class DataUnit

Data Members: No public data members are declared in this class.

The DataUnit class is a basic storage unit for data. Each instance of a DataUnit contains attribute information (defined by the DataUnit class definition) which includes a pointer to a chunk of memory allocated by the DataUnit for the storage of data.

The allocated storage provided by the DataUnit is typically allocated when the DataUnit is instantiated, but can also be reallocated when a value is assigned to a DataUnit. A DataUnit can be referenced (or shared) by a number of attributes or variables. Each DataUnit maintains a count of the number of attributes or variables that reference it.

When a `DataUnit` is created, its reference count is set to one. The count is incremented each time another value references it, and is decremented each time another value dereferences it. When the reference count goes to zero, the storage allocated for the `DataUnit` is deallocated. Each `DataUnit` also keeps track of the amount of dynamic memory it has allocated for storing data.

TABLE 2-20 lists the `DataUnit` class public functions.

TABLE 2-20 `DataUnit` Public Functions

Function Name	Description
<code>Octet*</code> <code>*</code> <code>[]</code> <code>=</code> <code>==</code> <code>!=</code>	Operator overloading
<code>size</code>	Description of the <code>DataUnit</code>
<code>cmp</code>	Compare two <code>DataUnits</code> for sorting
<code>copy</code> <code>catenate</code>	Create a new data unit from an existing one, or from two existing ones
<code>fragment</code> <code>copyin</code> <code>copyout</code>	Set or get a portion of the <code>DataUnit</code>
<code>chp</code>	Data as a character string

2.15.1 Constructors

This section describes the constructors for the `DataUnit` class.

```
DataUnit();
```

This constructor creates an instance of a `DataUnit`, but does not allocate any memory for storing data. The size of the `DataUnit` is set to zero.

```
DataUnit(U32 size, Octet *data = 0);
```


DataUnit Class

This constructor creates an instance of a `DataUnit` and allocates the number of bytes of memory specified by *size* for storing data. If the data specified by the `Octet *data` parameter is non-null, *size* bytes of data are copied from the octet string specified by *data* to the newly allocated storage, and the size of the `DataUnit` is set to *size*.

```
DataUnit(const char *str)
```

This constructor creates an instance of `DataUnit` from a NULL terminated string *str*.

Note – The `DataUnit` instance points to the same memory as *str* and it is assumed that the memory pointed by *str* is valid and is *not* changed during the lifetime of the `DataUnit` instance being created.

Unlike `DataUnit (char *str)`, this constructor does not allocate new memory and copy contents of *str* into it. (See `DataUnit(char *str)`)

Note – This constructor does not make a deep copy of the data passed to it. The correct way to use `RWCString.data()` to construct a `DataUnit` is to type cast `const char *` to `char *` so that `DataUnit` will keep its own copy of chars in `RWCString`, otherwise `DataUnit` will share the same data (chars) with `RWCString`. When `RWCString` goes out of scope, the data in `RWCString` gets destroyed, the `DataUnit` is still trying to use that data which has been destroyed by `RWCString`.

```
DataUnit(const DataUnit &du);
```

This constructor creates an instance of `DataUnit` but does not allocate any storage for it. The new `DataUnit` points at the storage allocated for the `DataUnit` identified by *du*. The reference count for the shared memory is incremented to reflect the additional `DataUnit` that is now pointing to it.

Note – The `DataUnit` instance points to the same memory as *du* and it is assumed that the memory pointed by *du* is valid and is *not* changed during the lifetime of the `DataUnit` instance being created.

```
DataUnit(U32 size, const Octet *data);
```

This constructor creates an instance of `DataUnit` which points at the storage associated with *data*. Its length is indicated by *size*.

Note – This constructor does not make a deep copy of the data passed to it. The correct way to use `RWCString.data()` to construct a `DataUnit` is to type cast `const char *` to `char *` so that `DataUnit` will keep its own copy of chars in `RWCString`, otherwise `DataUnit` will share the same data (chars) with `RWCString`. When `RWCString` goes out of scope, the data in `RWCString` gets destroyed, the `DataUnit` is still trying to use that data which has been destroyed by `RWCString`.

```
DataUnit(U32 size, int fill);
```

This constructor creates an instance of `DataUnit` and allocates storage of length *size* bytes. Each byte of the allocated storage is assigned the value *fill*.

```
DataUnit(const char *str);
```

This constructor creates an instance of `DataUnit` with storage sufficient to hold a duplicate of *str* (assumed to be null-terminated), which is copied to the `DataUnit`. The `DataUnit` points to the same area of memory as `const char *str`; this assumes that the string could change.

2.15.2 Destructor

```
~DataUnit();
```

This destructor decrements the reference count for the (potentially) shared memory region used to store data for the `DataUnit`. If the reference count reaches zero, the memory used to store data is deallocated.

2.15.3 DataUnit Operator Overloading

This sections describes the operators for the `DataUnit` class.

```
operator const Octet *() const;
```

DataUnit Class

This operator returns a pointer to the first Octet of allocated storage in the DataUnit.

```
const DataUnit &operator = (const DataUnit &du);
```

This operator returns a reference to the storage allocated for the DataUnit specified by *du*. The reference count for the storage is incremented. If the DataUnit (*this) previously pointed to allocated storage, the reference count for that storage is decremented, and if the reference count reaches zero, the storage is deallocated.

```
Octet &operator *();
```

```
const Octet &operator *() const;
```

This operator returns a reference to the first Octet of allocated storage in the DataUnit. This operator triggers an assertion failure if the DataUnit is uninitialized.

DataUnit Class

```
Octet &operator [] (U32 index);
```

```
const Octet &operator [] (U32 index) const;
```

These operators return a reference to the Octet of allocated storage that is *index* bytes past the first byte of storage allocated for the `DataUnit`. The first byte of allocated storage would have an index value of 0; the second byte, an index value of 1; the third byte, an index value of 2; and so on. This operator triggers an assertion failure if the value specified by the index parameter is greater than the size of the allocated storage, or if the `DataUnit` is uninitialized.

```
int operator == (const DataUnit &du) const
```

This operator compares the `DataUnit` specified by *du* to a previously defined `DataUnit`, and returns a nonzero integer if the two are equivalent. Similar functionality is found in the *cmp* member function.

```
int operator == (const char *str) const;
```

This operator compares the contents of the `DataUnit` with *str*. If they are the same, the function returns 1. If they differ, it returns 0.

```
int operator != (const DataUnit &du) const
```

This operator compares the `DataUnit` specified by *du* to a previously defined `DataUnit`, and returns a nonzero integer if the two are not equivalent. Similar functionality is found in the *cmp* member function.

```
int operator != (const char *str) const;
```

This operator compares the contents of the `DataUnit` with *str*. If they are the same, the function returns 0. If they differ, it returns 1.

2.15.4 DataUnit Member Functions

This sections describes the member functions of the DataUnit class.

catenate

```
friend DataUnit catenate(const DataUnit &du1,  
    const DataUnit &du2)
```

This function call constructs a DataUnit and allocates it dynamic storage equal to the size of the dynamic storage of *du1*, plus the size of the dynamic storage for *du2*. The `catenate` function then copies the allocated storage from *du1*, followed by the storage from *du2*, to the newly allocated storage. The function then returns the new DataUnit.

chp

```
char* chp() const;
```

This function call returns a pointer to a character string containing a representation of the DataUnit's data. The representation does not have a trailing newline character.

cmp

```
friend int cmp(const DataUnit &du1, const DataUnit &du2)
```

This function call performs a machine-dependent comparison of the two DataUnit's specified by *du1* and *du2*. The function returns a result in the same way as `strcmp`: that is, the result is 0 if the two DataUnit's compare equal, -1 if *du1* is lexicographically less than *du2*, and 1 if *du1* is lexicographically greater than *du2*.

DataUnit Class

copy

```
void copy (U32 pos
           U32 ln,
           const DataUnit &du,
           U32 du_pos)
```

This function call copies data from the `DataUnit` specified by *du* to the `DataUnit` specified by **this*. Data is copied from *du* starting from an offset into *du*'s allocated storage specified by *du_pos*. The data is copied to **this* starting at an offset into **this*'s allocated storage specified by *pos*. The number of octets copied is specified by *ln*. This function triggers an assertion failure if the number of octets to copy, specified by *ln*, plus the starting point specified by either *du_pos* or *pos*, is greater than the storage allocated for either *du* or **this*, respectively.

copy

```
DataUnit copy()
```

This function copies one `DataUnit` structure to another `DataUnit` structure.

copyin

```
void copyin (const Octet *const data, U32 pos, U32 ln);
```

This function call copies data from the location specified by the `Octet *data` parameter, into the dynamic memory allocated for the `DataUnit`. Data is copied into the allocated storage starting at a point that is *pos* bytes past the start of the allocated storage. The number of octets to copy is specified by *ln*. The function triggers an assertion failure if the starting position (specified by *pos*), plus the length to copy (specified by *ln*) is greater than the number of octets of storage allocated for the `DataUnit`.

DataUnit Class

copyout

```
void copyout (Octet * const data
              U32 pos,
              U32 ln) const;
```

This function call copies data from the `DataUnit` to location specified by the `Octet *data` parameter. Data is copied from the allocated storage of the `DataUnit` starting at a point that is *pos* bytes past the start of the allocated storage. The number of octets to copy is specified by *ln*. The function triggers an assertion failure if the starting position (specified by *pos*), plus the length to copy (specified by *ln*) is greater than the number of octets of storage allocated for the `DataUnit`.

equiv

```
static int equiv (Ptr n1,Ptr n2);
```

This function call is used to compare two `DataUnit` pointers in a manner consistent with the declaration of the Hash macros. The result is zero if the `DataUnit` pointed by *n1* is equal to the `DataUnit` pointed by *n2*, otherwise no zero.

fragment

```
DataUnit fragment(U32 st,U32 ln) const;
```

This function call returns a `DataUnit` that points to a portion (or fragment) of the dynamic storage allocated for the `DataUnit` specified by **this*. The fragment shares data with a portion of the storage allocated for the original `DataUnit`. The fragment starts *st* octets after the start of the allocated storage and extends for *ln* octets.

This function triggers an assertion failure either of the following occur:

- The start of the fragment, specified by *st*, plus the length of the fragment, specified by *ln*, is greater than the length of the allocated storage of `DataUnit` specified by **this*.
- The `DataUnit`, specified by **this*, is not initialized but *st* or *ln* is nonzero.

DataUnit Class

hash

```
static U32 hash(DataUnit *du)
```

This function call can be used when declaring Hash macros where a DataUnit is used as the key.

print

```
static DataUnit printf(const char *format,...);
```

```
void print(FILE *fp) const;
```

```
void print(Debug &deb = misc_stdout) const;
```

```
void print(char *c) const;
```

Each of the preceding function calls prints a formatted record of the DUclass, appending it either to the file whose pointer is *fp*, or the debug stream *deb*, or to the character string pointed to by *c*.

size

```
U32 size() const
```

This function call returns the number of Octets of the storage allocated for the DataUnit.

unshare

```
void unshare()
```

This function call makes memory storage associated with a DataUnit not shared. Creates a private copy if it is currently shared.

2.16 Dictionary Class

Declared in: `dict.hh`

The `Dictionary` class is the set of facilities that are provided to C++ classes created using the `Dictionarydeclare` macro. Dictionary tables are accessed in two ways: as indexable elements and using a key/value paradigm. Implicit in the definition of the dictionary is the `lookup()` function that provides the mechanism to map an arbitrary key value to an index in the array.

```
#define Dictionary (K,T)
    name3(K,T,Dict)
#define Dictionarydeclare(K,T)
```

TABLE 2-21 Dictionary Protected Variables

<code>const T *table_</code>	The internal table of elements
<code>U32 len_</code>	The number of elements in the table

TABLE 2-22 Dictionary Functions

<code>lookup</code>	Look up an element
<code>num_elems</code>	Return the number of elements
<code>position</code>	Return the index of an element
<code>table</code>	Return a pointer to the table
<code>[]</code>	Operator overloading

2.16.1 Constructor

```
Dictionary(K,T)(const T *t,U32 ne)
```

This constructor initializes a table pointer to *t* and assigns a length of *ne*.

2.16.2 Dictionary Operator Overloading

```
const T&operator[](U32 pos) const
```

This function returns the n th element of the table where pos equals n .

2.16.3 Dictionary Member Functions

lookup

```
const T*lookup(const K &key) const
```

The `lookup` function, which must be implemented for each K and T that are defined for a usage of `Dictionarydeclare()`, returns a pointer to a table element of type T based on a key of type K .

num_elems

```
U32 num_elems() const
```

The `num_elems` function returns the number of elements in the table.

position

```
U32 position(const T *t) const
```

Given a pointer to the table element t , this function returns the index to the table.

GenInt Class

table

```
const T *table() const
```

This function returns a pointer to the table of elements managed within the Dictionary object.

2.17 GenInt Class

Declared in: `genint.hh`

The `GenInt` class is used to represent integers of arbitrary size.

2.17.1 Constructors

```
GenInt()
```

`GenInt()` is the default constructor.

```
GenInt(int ival)
```

Constructs an instance of `GenInt` with initial value *ival*.

```
GenInt(I32 ival)
```

Constructs an instance of `GenInt` with initial value *ival*.

```
GenInt(U32 uval)
```

Constructs an instance of `GenInt` with initial value *uval*.

```
GenInt(const char *cval)
```

GenInt Class

Constructs an instance of GenInt given the human readable string format in two's complement form.

```
GenInt(const DataUnit &d)
```

Constructs an instance of GenInt given a DataUnit.

2.17.2 Copy Constructor

```
GenInt(const GenInt &val)
```

2.17.3 GenInt Member Functions

sign

```
I32 sign() const;
```

The `sign` function returns the sign of the integer.

bits

```
U32 bits() const;
```

The `bits` function returns the number of bits in the integer.

size

```
U32 size() const;
```

The `size` function returns the storage size, in bytes, of the integer.

GenInt Class

div

```
GetInt div(const GenInt &v, GenInt &remainder) const;
```

The `div` function invokes the division operation.

format

```
U32 format(char *buf, U32 buf_len) const;
```

The `format` function converts an integer to a character string.

encode

```
DataUnit encode() const;
```

The `encode` function returns the two's complement form of the integer.

operator double

```
operator double() const;
```

The `operator double` function returns the double value of `GenInt`.

operator I32

```
operator I32() const;
```

The `operator I32` function returns the I32 value of `GenInt`. If `GenInt` is larger, it returns only the four most significant bytes.

GenInt Class

operator U32

```
operator U32() const;
```

The `operator U32` function returns the U32 value of `GenInt`. If `GenInt` is larger, it returns only the four most significant bytes.

operator +

```
operator +() const;
```

Unary + operator.

```
operator +(const GenInt &v) const;
```

Addition operation.

&operator +=

```
&operator +=(const GenInt &v)
```

The compound additive assignment operator.

operator -

```
operator -() const;
```

Unary - operator.

```
operator -(const GenInt &v) const;
```

The subtraction operation.

GenInt Class

`&operator -=`

```
&operator -=(const GenInt &v) {  
    *this = *this - v;  
    return *this;  
};
```

The compound subtractive assignment operator.

`operator !`

```
operator !() const;
```

Unary boolean operator. Returns TRUE if value is 0, FALSE otherwise.

`operator *`

```
operator *(const GenInt &v) const;
```

The `operator *` function multiplies two `GenInt`s.

`&operator *=`

```
&operator *=(const GenInt &v)
```

The compound multiplicative assignment operator.

`operator /`

```
operator /(const GenInt &v) const;
```

The division operation.

GenInt Class

`&operator /=`

```
&operator /=(const GenInt &v)
```

The compound division assignment operator.

`operator %`

```
operator %(const GenInt &v) const;
```

The mod operation.

`&operator %=`

```
&operator %=(const GenInt &v)
```

The compound mod assignment operator.

`operator <`

```
operator <(const GenInt &v) const;
```

The “less than” operator.

`operator >`

```
operator >(const GenInt &v) const;
```

The “greater than” operator.

GenInt Class

operator <=

```
operator <=(const GenInt &v) const;
```

The “less than or equal” operator.

operator ==

```
operator ==(const GenInt &v) const;
```

The “greater than or equal” operator.

operator &

```
operator &(const GenInt &v) const;
```

The bitwise AND operator.

&operator &=

```
&operator &=(const GenInt &v)
```

The compound bitwise AND assignment operator.

operator |

```
operator |(const GenInt &v) const;
```

The bitwise OR operator.

GenInt Class

`&operator |=`

```
&operator |=(const GenInt &v)
```

The compound bitwise OR assignment operator.

`operator ^`

```
operator ^(const GenInt &v) const;
```

The bitwise XOR operator.

`&operator ^=`

```
&operator ^=(const GenInt &v)
```

The compound bitwise XOR assignment operator.

`operator ~`

```
operator ~() const;
```

The bitwise NOT operator.

`operator ==`

```
operator ==(const GenInt &v) const;
```

The binary equality operator.

Hash Class

operator !=

```
operator !=(const GenInt &v) const;
```

The binary not equal operator.

2.18 Hash Class

Inheritance: class HashImpl

Declared in: hash.hh

2.18.1 Hash Member Functions

fetch

```
valtype *fetch(const keytype *key) const
```

or

```
valtype *fetch(const keytype *key, U32 hashval) const
```

This function fetches an element with the specified keytype or with specified keytype and hashval from the hash table.

store

```
void store(const keytype *key, valtype *val)
```

or

```
void store(const keytype *key, valtype *val, U32 hashval)
```

Hash Class

This function stores an element with the specified keytype and valtype or with the specified keytype, valtype, and hashval into the hash table.

destroy

```
void destroy(const keytype *key)
```

or

```
void destroy(const keytype *key, U32 hashval)
```

This function destroys an element with the specified keytype or with the specified keytype and hashval from the hash table.

iterate

```
void iterate()
```

This function iterates the hash table.

next

```
Boolean next(keytype* &key, valtype* &val, U32 &hashval)
```

This function checks whether there is an element after the element with the specified keytype, valtype, and hashval from the hash table. If there is one, it returns TRUE, otherwise FALSE.

2.19 Hashdeclare Macro

Declared in: `hash.hh`

```
Hashdeclare(keytype, valtype, hashfun, eqfun, delkey, delval)
```

`Hashdeclare` is a macro used to create a hash table class that is specific to the types of its keys and values. To refer to the class thus created, use the `Hash(keytype, valtype)` macro (described below).

The arguments *keytype* and *valtype* are the names of declared types or classes. The arguments *hashfun*, *eqfun*, *delkey*, and *delval* are the names of functions required by the constructor for the `HashImpl` class.

Because the macro invokes functions defined in the `HashImpl` class, the class it produces is in a sense “derived” from `HashImpl`. Each of the member functions defined by `Hashdeclare` has a corresponding function of the same name defined by the `HashImpl` class. The macro and the `HashImpl` class also provide alternate calls to calculate the hashed values for you when you don’t want to do it yourself.

2.20 HashImpl Class

Inheritance: `class HashImpl`

Declared in: `hash.hh`

The `HashImpl` class provides a template class used by the `Hashdeclare()` macro to implement a dynamically growing hash table. It maintains the private data structures that constitute the hash table.

The `HashImpl` class is intended to be somewhat primitive; it is normally hidden behind the `HashDeclare` macro. The `HashImpl` member functions have no idea what they’re actually managing. All keys and values are passed to them as `void*` values.

For this reason, when the constructor for `HashImpl` is called, you must supply pointers to three functions:

- A function that takes two arguments of type `void*` and returns whether the two values are to be considered equal.

- A function (which might be `NULL`) to invoke when keys are to be destroyed. It is passed a single `void*` value.
- A function (which might also be `NULL`) to invoke when values are to be destroyed. It is also passed a single `void*` value.

These last two functions, if specified, are called whenever key/value pairs need to be deleted from the hash table, including when the entire table is destroyed.

It is often the case that the same key is used several times in a row. For this reason, all of the member functions that take a key argument also take an argument that is the hashed value of the key. The hash value needs to be recomputed only when the key changes. You can call any hash function you like, but it should return an unsigned 32-bit integer, hopefully with a uniformly pseudorandom distribution in the lowest n bits (where n is the number of bits necessary to index into a hash table big enough to hold the maximum number of entries, more or less). A sample hash function for null-terminated strings is supplied; see the member function, `strhash()`.

In order for a key to match an entry in the table, its hashed value must first match the hashed value stored in the table entry. You could conceivably make use of this in a smart hash function to make otherwise identical keys behave as if they're different keys.

Variables: No public variables declared in this class.

TABLE 2-23 HashImpl Public Functions

Public	
Fetch a value from the hash table	<code>fetch</code> <code>next</code>
Store a key and value in the hash table	<code>store</code>
Destroy a key-value pair	<code>destroy</code>

2.20.1 Constructor

```
HashImpl(int (*eqfun)(Ptr, Ptr),  
         void (*delkey)(Ptr),  
         void (*delval)(Ptr))
```

2.20.2 Destructor

```
~HashImpl()
```

2.20.3 HashImpl Member Functions

Apart from `iterate()`, the member functions of `HashImpl` are provided as implementations of the member functions of the specific hash class created by a use of the `Hashdeclare` macro.

`clear`

```
void clear()
```

The preceding function call nulls out the class structure,

`destroy`

```
void destroy(Ptr key, U32 hashval)
```

Deletes the key/value pair from the hash table. Returns the deleted value (which might have been destroyed, if a *delval* function was specified to the constructor).

HashImpl Class

fetch

```
Ptr fetch(const Ptr key, U32 hashval) const
```

Returns a pointer to the value associated with *key* and *hashval*.

iterate

```
void iterate()
```

Used internally by the class constructed by the `Hashdeclare` macro to reset the hash table's built-in iterator to the beginning.

next

```
Boolean next(Ptr &key,  
             Ptr &val,  
             U32 &hashval)
```

Returns `TRUE` if more entries remain in the hash table. Sets pointer (`Ptr`) variables for the next key/value pair from the hash table, according to the table's built-in iterator. It also sets the corresponding *hashval* variable. It is legal to destroy the current value.

store

```
void store(Ptr key,  
           U32 hashval,  
           Ptr value);
```

Stores a pointer to the value associated with the specified *key* and *hashval*, and returns that value. Any previous value associated with that *key* and *hashval* are destroyed. Returns the new value.

Hdict Class

strhash

Declared in: hash.hh

```
U32 strhash(const char *str)
```

The `strhash()` function is a sample hashing function for null-terminated strings. It could be used to calculate hash values for passing to the member functions of `HashImpl`, or its name could be passed to the `Hashdeclare` macro as the hash function for keys of the appropriate type.

2.21 Hdict Class

Declared in: dict.hh

The `Hdict` class is the set of facilities provided to C++ classes created using the `Hdictdeclare()` macro. It utilizes the Hash mechanism to manage the lookup of elements in its table. It assumes that an appropriate `Arraydeclare(T)` and `Hashdeclare(K,T)` have preceded its declaration.

```
#define Hdict(K,T)
    name3(K,T,Hdcit)
#define Hdictdeclare(K,T)
```

TABLE 2-24 Hdict Protected Variables

<code>Hash(K,T) *_hash;</code>	The internal hash table of elements
<code>Array(T) _array;</code>	The array of elements

TABLE 2-25 Hdict Public Functions

<code>lookup</code>	Lookup an element
<code>num_elems</code>	Return the number of elements
<code>table</code>	Return a pointer to the table
<code>position</code>	Return the index of an element
<code>set</code>	Initialize the array and the hash table

2.21.1 Constructors

```
Hdict(K,T)
```

This constructor creates an empty table.

```
Hdict(K,T)(Array(T)&a)
```

This constructor creates a hashing dictionary by stealing the array associated with *a* and initializing an internal hash table.

2.21.2 Hdict (K,T) Operator Overloading

```
const T&operator[](U32 pos) const
```

This function returns the *n*th element of the table where *n* equals *pos*.

2.21.3 Hdict Member Functions

lookup

```
const T*lookup(K *k) const
```

The `lookup` function, which must be implemented for each *K* and *T* that are defined for a usage of `Hdictdeclare()`, returns a pointer to a table element of type *T* based on a key of type *K*.

num_elems

```
U32 num_elems() const
```

The `num_elems` function returns the number of elements in the table.

Hrefdict Class

position

```
U32 position(const T *t) const
```

Given a pointer to the table element *t*, the `position` function returns the index to the table.

table

```
const T *table() const
```

The `table` function returns a pointer to the table of elements managed within the Hdict Dictionary.

set

```
void set(Array(T)&a)
```

The `set` function allows you to populate your Hdict Dictionary with the specified array.

2.22 Hrefdict Class

Declared in: `dict.hh`

The `Hrefdict` class is the set of facilities provided to C++ classes created using the `Hrefdictdeclare()` macro. It utilizes the Hash mechanism to manage the lookup of elements in its table. It assumes that an appropriate `Arraydeclare(T)` and `Hashdeclare(K,T)` have preceded its declaration.

```
#define Hrefdict(K,T)  
    name3(K,T,Hrefdict)  
#define Hrefdictdeclare(K,T)
```

Hrefdict Class

TABLE 2-25 lists the Hrefdict protected variables.

TABLE 2-26 Hrefdict Protected Variables

Hash(K,T) *_hash;	The internal hash table of elements
const T *table_	The internal table of elements
U32 len_	The number of elements in the table

TABLE 2-27 lists the Hrefdict public functions.

TABLE 2-27 Hrefdict Public Functions

lookup	Lookup an element
num_elems	Return the number of elements
table	Return a pointer to the table
position	Return the index of an element
set	Initialize the hash table

2.22.1 Constructors

Hrefdict(K,T)

This constructor creates an empty table.

Hrefdict(K,T)(const T *t,const U32 ne)

This constructor initializes a table pointer to *t* and assigns a length of *ne*.

2.22.2 Hrefdict (K, T) Operator Overloading

const T&operator[](U32 pos) const

This function returns the *n*th element of the table where *n* equals *pos*.

2.22.3 Hrefdict Member Functions

lookup

```
const T*lookup(K *k) const
```

The `lookup` function, which must be implemented for each *K* and *T* that are defined for a usage of `Hrefdictdeclare()`, returns a pointer to a table element of type *T* based on a key of type *K*.

num_elems

```
U32 num_elems() const
```

The `num_elems` function returns the number of elements in the table.

position

```
U32 position(const T *t) const
```

Given a pointer to the table element *t*, the `position` function returns the index to the table.

table

```
const T *table() const
```

The `table` function returns a pointer to the table of elements managed within the Dictionary.

Oid Class

set

void set()

The `set` function allows you to populate your Hrefdict Dictionary with the table specified in the constructor.

2.23

Oid Class

Inheritance: `class Oid : public DataUnit`

Data Members: No public data members declared in this class

`#include <pmi/oid.hh>`

The `Oid` class defines a container for an object identifier. Each object identifier is a sequence of numbers that identify an object by an integer denoting its assigned position at each level of a tree-structured registry of object names.

Many of the properties of the `Oid` implementation (for example, data storage, sharing) are derived from `DataUnit`. TABLE 2-28 lists the `Oid` class public functions.

TABLE 2-28 Oid Public Functions

Function Name	Descriptions
<code>=</code>	Operator overloading
<code>copy_oid</code> <code>format</code> <code>print</code>	Extract the entire oid
<code>num_ids</code>	Length of the oid
<code>get_id</code>	Extract a given id from within the oid
<code>add_id</code> <code>add_last_id</code> <code>append</code>	Add to the oid
<code>is_same_prefix</code>	Compare two ids values

2.23.1 Constructors

This section describes the constructors of the `Oid` class.

```
Oid()
```

This constructor constructs an empty `Oid`.

```
Oid(U32 size, const Octet *data = 0)
```

```
Oid(const DataUnit &du)
```

```
Oid(const Oid &oid)
```

```
Oid(const char *str)
```

Each of the preceding constructors construct an `Oid` from a particular representation of the list of IDs that comprise it.

```
Oid(U32 size,  
    U32 first_id,  
    U32 second_id,  
    U32 &place)
```

This constructor constructs an object identifier whose first identifier is *first_id* and whose second identifier is *second_id*. The size of the allocated memory is *size*. The returned value *place* is used when appending new identifiers to the object identifier string (see `add_id`).

2.23.2 Oid Operator Overloading

This section describes the operators for the `Oid` class.

```
const Oid &operator = (const Oid &oid)
```

This operator shares the data storage associated with *oid*.

2.23.3 Oid Member Functions

This section describes the member functions of the `Oid` class.

`add_id`

```
Result add_id(U32 id,U32 &place)
```

This function call appends *id* to the `Oid`, and sets *place* to the current size (and hence the position at which *id* was inserted).

The `Oid(U32, U32, U32, U32)` constructor initializes *place*.

If need be, `add_id()` extends the `Oid` to accommodate the new *id*. (You could use the `Oid(size)` constructor to pre-extend the `Oid` instance to the desired size.)

`add_last_id`

```
Result add_last_id(U32 id,U32 &place)
```

This function call appends *id* to the last id in the `Oid` instance. Any additional storage is released.

`append`

```
static Result append(Oid &front,  
                    Oid &back,  
                    U32 front_place)
```

This function call modifies *front* by appending the IDs contained in *back*. Returns OK if this is completed. Sets *front_place* to the last-written position in *front* (which is also the numbers of ids in *front* when the operation is complete).

Oid Class

copy_oid

```
static Result copy_oid(const Oid &src, Oid &dest)
```

This function call makes *dest* the same as *src*.

format

```
Result format(char *buf, U32 buf_len) const
```

This function call writes to *buf* a string of length *buf_len* containing the formatted representation of the Oid. In the string, each ID is represented as decimal digits, and successive IDs are separated by a dot.

get_id

```
Result get_id(U32 id_num,  
              U32 &id,  
              U32 &place) const
```

This function call extracts from the Oid its *id_num*'th id and stores it in *id*. (The first *id* is considered to be at position 1). To speed up subsequent access, stores at *place* the current position within the Oid, and initialize it to zero before the first use.

is_same_prefix

```
static Boolean is_same_prefix(const Oid &o1, const Oid &o2)
```

This function call compares two Oids to determine whether the one with the shorter sequence of IDs is a prefix of the other, and returns `TRUE` if it is. (If both Oids contain the same number of IDs, this boils down to asking whether they are equal.)

This function returns `FALSE` if the two Oids do not have the same prefix.

Asn1TypeDefinedType Declarations

num_ids

```
U32 num_ids() const
```

This function call returns the number of IDs in the `Oid` instance.

print

```
void print(FILE *fp) const;
```

```
void print(Debug &deb = misc_stdout) const;
```

```
void print(char *c) const;
```

Each of the preceding function calls print a formatted record of the `Oid`, appending it either to the file whose pointer is *fp*, or the debug stream *deb*, or to the character string pointed to by *c*.

2.24 Asn1TypeDefinedType Declarations

Following is a list of the `Asn1TypeDefinedType` declarations:

```
extern Asn1TypeDefinedType NumericStringType;  
extern Asn1TypeDefinedType PrintableStringType;  
extern Asn1TypeDefinedType TeletexStringType;  
extern Asn1TypeDefinedType VideotexStringType;  
extern Asn1TypeDefinedType VisibleStringType;  
extern Asn1TypeDefinedType IA5StringType;  
extern Asn1TypeDefinedType GraphicStringType;  
extern Asn1TypeDefinedType GeneralStringType;  
extern Asn1TypeDefinedType GeneralizedTimeType;  
extern Asn1TypeDefinedType UTCTimeType;  
extern Asn1TypeDefinedType EXTERNALType;  
extern Asn1TypeDefinedType ObjectDescriptorType;
```

2.24.1 Asn1SubTypeKind

Following is the Asn1SubTypeKind declaration:

```
enum Asn1SubTypeKind
{
    ASK_NONE,
    ASK_SINGLE,
    ASK_CONTAINED,
    ASK_RANGE,
    ASK_PERMITTED,
    ASK_SIZE,
    ASK_INNER_SINGLE,
    ASK_INNER_MULTIPLE
} ;
```

2.24.2 Asn1SubTypeSize

Following is the Asn1SubTypeSize declaration:

```
typedef Asn1SubTypeSize Asn1SubTypePermitted;
typedef Asn1SubTypeSize Asn1SubTypeInnerSingle;
```

2.24.3 Asn1Kind

Following is the Asn1Kind declaration:

```
enum Asn1Kind {  
    AK_NONE,  
    AK_BOOLEAN,  
    AK_INTEGER,  
    AK_BIT_STRING,  
    AK_OCTET_STRING,  
    AK_NULL,  
    AK_SEQUENCE,  
    AK_SEQUENCE_OF,  
    AK_SET,  
    AK_SET_OF,  
    AK_CHOICE,  
    AK_SELECTION,  
    AK_TAGGED,  
    AK_ANY,  
    AK_OBJECT_IDENTIFIER,  
    AK_ENUMERATED,  
    AK_REAL,  
    AK_SUBTYPE,  
    AK_DEFINED_TYPE  
};
```

2.24.4 Asn1TypeE

Following is the Asn1TypeE declaration:

```
typedef Asn1TypeE Asn1TypeSeqOf;  
typedef Asn1TypeE Asn1TypeSetOf;
```

2.24.5 Asn1TypeEL

Following is the Asn1TypeEL declaration:

```
typedef Asn1TypeEL Asn1TypeSeq;  
typedef Asn1TypeEL Asn1TypeSet;
```

2.24.6 Asn1TypeNN

Following is the Asn1TypeNN declaration:

```
typedef Asn1TypeNN Asn1TypeInt;
typedef Asn1TypeNN Asn1TypeEnum;
typedef Asn1TypeNN Asn1TypeBitStr;
```

2.24.7 Asn1TagClass

Following is the Asn1TagClass declaration:

```
declared in: /opt/SUNWconn/em/include/pmi/asn1_val.hh
typedef enum Asn1TagClass
{
    CLASS_UNIV,
    CLASS_APPL,
    CLASS_CONT,
    CLASS_PRIV} ;
```

2.24.8 Asn1Tagging

Following is the Asn1Tagging declaration:

```
// declared in: /opt/SUNWconn/em/include/pmi/asn1_val.hh
typedef enum Asn1Tagging
{
    TAG_EXPLICIT,
    TAG_IMPLICIT,
} ;
```

2.25 Queue Class

Inheritance: class QueueImpl

Declared in: queue.hh

2.25.1 Queue Member Functions

enq

```
type *enq(type *qe)
```

This function inserts an element at the end of a queue with the specified type.

prq

```
type *prq(type *qe)
```

This function inserts an element at the head of a queue with the specified type.

inq

```
type *inq(type *oqe, type *nge)
```

This function inserts the element `nge` into a queue with the specified type before the element `oqe`.

apq

```
type *apq(Queue (type) &q)
```

This function appends a queue to a queue with the specified type.

ppq

```
type *ppq(Queue (type) &q)
```

This function prepends a queue to a queue with the specified type.

Queue Class

deq

```
type *deq()
```

This function deletes the first element in a queue with the specified type.

rnq

```
type *rnq()
```

This function rotates to the next element in a queue with the specified type.

rpq

```
type *rpq()
```

This function rotates to the previous element in a queue with the specified type.

fiq

```
type *fiq() const
```

This function returns the first element in a queue with the specified type.

liq

```
type *liq() const
```

This function returns the last element in a queue with the specified type.

Queuedeclare Macro

exq

```
type *exq(type *qe)
```

This function deletes an element in a queue with the specified type.

niq

```
type *niq(type *qe) const
```

This function finds the next element in a queue with the specified type.

piq

```
type *piq(type *qe) const
```

This function finds the previous element in a queue with the specified type.

nmq

```
U32 nmq() const
```

This function returns the number of elements in a queue with the specified type.

2.26 Queuedeclare Macro

Declared in: `queue.hh`

```
Queuedeclare(type)
```

The `Queuedeclare` macro is used to create a queue class with the specified type. Please see `class queue(type)` for more detail.

2.27 Timer Class

Inheritance: class Timer

```
#include <pmi/gsched.hh>
```

An event that is scheduled to occur after a specific interval is represented as an instance of the Timer class. Timer events are posted or removed from the timer queue by the functions `post_timer()`, `purge_timer()`, `purge_timer_data()`, and `purge_timer_handler()`. Each function is declared in `sched.hh`.

A Timer event is not dispatched before the specified interval has elapsed. However, it might have to wait longer if other processing has to happen when its interval has elapsed.

When the callback does occur, the timer automatically reposts itself if you specified a *reload time* in the Timer. (By default, the reload time is 0, meaning no reposting.) Both the invocation time and the reload time are specified in milliseconds.

The invocation time is the interval from the time at which you call the `post_timer()` routine. The reload time is the interval from the originally scheduled time (not from the time at which the callback was actually dispatched).

If the callback is delayed to the point that the reload interval has already elapsed, the scheduler skips a reload for an interval that has now elapsed, and schedules the reload for the next upcoming multiple of the reload interval.

Timers only guarantee that the callback does not run too early. There is no guarantee about it running too late, so don't try to use this to control real-time interactions.

Timers can be purged from the timer queue by matching on the enqueued object's handler, data, or both.

2.27.1 Default Constructor

```
Timer()
```

Resets the time and reloads the timer.

2.27.2 Constructor

```
Timer(MTime t, MTime re, CallbackHandler hand, Ptr d)
```

Resets the time and reloads the timer. This constructor has four arguments. The time (*t*) between now and the first callback; the reload time (*re*) between callbacks; the callback pointer (*hand*) to the functions which is executed; and the pointer (*d*) to the user data which is passed to the callback hand.

2.27.3 Operator

```
friend int operator==(const Timer &t1, const Timer &t2)
```

Compares the timers to determine whether they have the same callback function or not.

2.27.4 Related Global Functions

post_timer

```
void post_timer(const Timer&)
```

Posts a timer event into the timer queue.

post_timer_handler

```
void post_timer_handler(CallbackHandler handler);
```

Posts from the timer queue any timer events whose handler matches the specified *handler*.

Timer Class

`purge_timer`

```
void purge_timer(const Timer &)
```

Purges from the timer queue any matching timer events.

`purge_timer_data`

```
void purge_timer_data(Ptr data);
```

Purges from the timer queue any timer events whose data matches *data*.

`purge_timer_handler`

```
void purge_timer_handler(CallbackHandler handler);
```

Purges from the timer queue any timer events whose handler matches *handler*

`getGeneralizedTime`

```
void getGeneralizedTime();
```

The `getGeneralizedTime` function creates an ASN.1 encoded generalized time value for the current time.

High-Level PMI

The Solstice EM product provides a Portable Management Interface (PMI) with a suite of classes and member functions that provide effective access to the Solstice EM Management Information Server (MIS) without requiring detailed specification of the underlying MIS or mechanism. For most applications, the high-level usage of the PMI is sufficient for all interactions with the Solstice EM MIS.

This chapter comprises the following topics:

- Section 3.1 “Design Objectives” on page 3-1
- Section 3.2 “Object Management Model” on page 3-2
- Section 3.3 “Meta Data Repository” on page 3-7
- Section 3.4 “Symbolic Constants” on page 3-18
- Section 3.5 “Defined Types” on page 3-21
- Section 3.6 “Error Handling and Event Dispatching” on page 3-24
- Section 3.8 “High-Level PMI Classes” on page 3-26

3.1 Design Objectives

The PMI seeks to balance two contrasting goals:

- *Location transparency.* It should be convenient to write an application without having to know where and how its objects are stored.
- *Location flexibility.* It should be convenient to write an application that takes specific account of where and how its objects are stored.

To achieve location transparency, you need:

- MIS independence
- Automatic propagation of *event sieves* from the application to the MIS
- Automatic *caching* of various sorts of data in the application process

To achieve location flexibility, you need to:

- Access (some of) the low level primitives upon which the high-level usage of the PMI is built
- Use the low level primitives of the PMI in a way that does not confuse the high-level primitives of the PMI

3.2 Object Management Model

Low-level routines for manipulating objects tend to manage objects by sending them messages and waiting for replies. In Common Management Information Service (CMIS), for instance, there are messages to create and delete objects, to get and set attributes, and to perform various actions or to signal events.

The PMI replaces these notions with a model in which objects appear (as far as possible) to be local. The remote object is tracked in the application process by a local class object called an image, which acts as a surrogate for the remote object and tracks where it is and what it is doing.

3.2.1 Naming Objects

Objects are named by starting from a known starting point in a tree and traversing a map of containment relationships. The object's name is formed by concatenating the "key" for each of the steps in the traversal, with slashes between the components. To find an object's container, you have only to strip off the final component of the object's name.

An object name that begins without a slash is a local distinguished name and is under the local root. In the case of the Solstice EM MIS, the local root is `/systemID`. An object name that begins with a slash is a distinguished name, and refers to an object that is global (in the literal sense). Names used in the PMI can comprise a superset of the OSI naming tree. Names that are not part of the OSI naming tree must not conflict with those on the OSI naming tree.

Within the application, any object can also have a nickname. Nicknames offer a convenient way for you to program readably, and also provide a level of indirection that helps in the quest for abstraction.

3.2.2 Relationships Between Objects

The relationships between objects are represented using album objects.

You can think of undirected sibling-like relationships as a form of set membership. The album contains a set of `images`, so membership of an image in an album can model the inclusion of objects in a mathematical set. The application can build up an album by enumerating the `images` that it wants to include.

Other relationships are of a directed nature. The PMI models these relationships as albums built by a derivational rule rather than by enumeration. A simple derivation might form the set of objects that are “children” of another object. A more complex derivation might examine all the objects in another album, select a subset of them, and for each of that subset specify a relational attribute that defines a new set of objects.

3.2.3 Managing Notifications

An image (or an album of `images`) can alert the application when a change of state in which the application has expressed an interest occurs. The PMI transmits the event to the application by invoking the callback function that the application registered earlier.

3.2.4 Managing Data Types

Each image knows the attributes that a given object class supports. It also knows the type of each attribute. That lets the application deal with the object on a purely textual basis if it chooses.

The language in which textual data is expressed looks much like ASN.1 textual data. Scoping is indicated by curly braces, choice names are delimited by a colon, and so on. This approach was decided upon because ASN.1 specification is complete and mature, at least when compared with other abstract syntax notations. You can represent other abstract notations with this data language, but it maps most easily onto ASN.1.

Sometimes the application needs to deal with data apart from the definition of any particular object. While this can be done using text, as mentioned above, it is sometimes more convenient to pass around encoded data. The PMI supports the notion of typed, encoded data. Such an object is called a `Morf`. You can think of a `Morf` as a bare attribute without any associated object. It knows its type and the associated syntax, so you can convert its value to and from textual representation, as you would an attribute of an image object.

3.2.5 Object Schema Management

The application program can discover various facts about the object class and its various attributes by using the `get_prop()` and `get_attr_prop()` functions. These routines work much like the UNIX `getenv` call, but acquire their information from the MIS or from some associated repository of metadata. To avoid confusing these facts with the ordinary attributes contained in a managed object instance, these attribute-like facts are called "properties."

Properties occur among attributes used by the `Image` class, and in the arguments or results of methods of the `Image`, `Album`, and `Platform` classes. TABLE 3-1 summarizes where various properties occur. See also the descriptions of:

- `get_prop`, under the description of the `Album` class
- `get_prop`, under the description of the `Image` class
- `get_attr_prop`, also under the description of the `Image` class

TABLE 3-1 Properties in `Album`, `Image`, and `Platform`

Property	Image Attribute	Image	Album	Platform
ACCESS			X	
APPLICATION_INSTANCE_NAME				X
APPLICATION_TYPE				X
AUTOIMAGE			X	
DEFAULT_ALLOWED	X			
DEFAULT_TIMEOUT				X
DERIVATION			X	
EXCLUDE_ALLOWED	X			
EXISTS	X	X		
IGNORE_ALLOWED	X			
LOCATION				X
MOD_PENDING	X			
MODIFIABLE	X			
NICKNAME		X	X	
NICKNAME_IS_PERMANENT		X		
OBJCLASS		X		
OBJNAME		X		

TABLE 3-1 Properties in Album, Image, and Platform (*Continued*)

Property	Image Attribute	Image	Album	Platform
OWNERSHIP			X	
PLATFORM_NICKNAME				X
PLATFORM_OBJNAME				X
PLATFORM_TYPE				X
REPLACE_ALLOWED	X			
STATE		X	X	X
TRACKMODE	X	X	X	

3.2.6 Filtering as an Aspect of Album Derivation

The Album function `set_derivation()` (or its more general form `set_prop`) can specify a derivation that includes a CMIS filter. The derivation specifies three items, in this order:

- Object name
- Scope
- Filter

A slash separates the scope from the object name. If there is a filter, a slash separates the scope from the filter.

3.2.6.1 Object Name

An object name is a distinguished name in slash form. The object name specifies the base object for scoping. This base object is not necessarily one of the objects in the album.

It is permissible to omit the object name. If you omit the object name, the system object is assumed. Also if you omit the object name, you should not insert the slash that would separate the name from the derivation. If you write `/ALL`, you are indicating an object name of `/` (indicating the system object), followed by the scope `ALL`.

3.2.6.2 Scope

The scope can be any of those shown in TABLE 3-2.

TABLE 3-2 Scoping Parameters

Parameter	Description
ALL	All descendants of named object, including object.
LV <i>n</i>	Level <i>n</i> descendants only. Children are at LV(1). Grandchildren are at LV(2). The base object is at LV(0).
TO <i>n</i>	All levels down to level <i>n</i> , including object
*	Short for LV(1), i.e. children only
* / *	Short for LV(2), i.e. grandchildren only.
* / * / *	Short for LV(3), i.e. great-grandchildren only.

If omitted, the scope defaults to the base object only. The asterisk forms are short for LV(*n*), not TO(*n*).

3.2.6.3 Filter

The filter is currently specified in raw ASN.1 format. The definition of the filter goes inside the parentheses that follow `CMISFilter`. This syntax makes `CMISFilter` look like a function. In fact, it is not a function; the use of parentheses is a convention for delimiting the filter definition.

Omitting the filter has the same effect as a filter whose definition is `TRUE`, meaning “include everything specified by the scope.” The definition of a legal `CMISFilter` is specified in `etc/asn1/x711.asn1`, which is a formalization of the X.711 standard.

As an example of filtering, to find all of the `OMNIPoint` logs under the system object, any of the expressions shown below would be a valid argument to `Album::set_derivation()`.

```
/systemId="mysys"/LV(1)/
CMISFilter(item:equality:{objectClass,log})
or
LV(1)/CMISFilter(item: equality: {objectClass, log})
or
*/CMISFilter(item:equality:{objectClass,log})
```

Meta Data Repository

Building on the previous examples, to get only logs that are enabled, filter on `operationalState`, using `and`, as shown below:

```
LV(1)/CMISFilter(  
  and: {  
    item: equality: {objectClass, log},  
    item: equality: {operationalState, enabled}  
  }  
)
```

For the PMI, newlines are allowed and might enhance readability. The newlines might not be accepted by a shell.

3.2.6.4 Operation of a Filtering Derivation

When a filtered album is derived, the filtering is done automatically by the platform, so you never see any callbacks for the objects that are bypassed by the filter. If the album's `TRACKMODE` is set to `TRACK`, the album is maintained on the basis of the filter. That is, if an attribute changes in a way that makes the value of the filter `TRUE` (when it has been `FALSE`) or `FALSE` (when it has been `TRUE`), the album is automatically updated so that the image of the object is included (or excluded, as appropriate).

If the album is not set to `TRACK`, you can perform another scoped and filtered `M-GET` by calling `derive()` again.

If you execute `all_destroy()` on an album that has a derivation and that album is in a `DOWN` state, or is an `AUTOIMAGE` album, then the request is optimized to do a single scoped `M-DELETE` using the scope and filter specified for the album. (Otherwise a separate `M-DELETE` is issued for each member of the album, as before.)

The other `ALL` operations are not yet optimized in this way, but you can get the effect of an optimized `all_boot()`. If you execute `derive()` on an `AUTOIMAGE` album, the initial scoped `M-GET` fetches all the attributes at that time, rather than issuing a subsequent `M-GET` for each image.

3.3 Meta Data Repository

The Meta Data Repository (MDR) is where descriptions of managed objects are stored. A description for every object known to the MIS is stored in the MDR. This data encompasses everything from the syntax required to refer to the attribute, to the

composition of an object package. The MDR is initialized and updated by using the GDMO and ASN.1 compilers. MDR supports the following actions to provide information about the objects.

3.3.1 `getAttribute` Action

```
getAttribute ' "GDMO DOCNAME":attrName '
```

This action provides information about an attribute. It gives information about which ASN.1 module the attribute is actually defined.

3.3.2 `getAllDocuments` Action

```
getAllDocuments getAllDocuments
```

This action provides a list of all documents. You still have to provide either a class name or an oid as the argument.

3.3.3 `getAsn1Module` Action

```
getAsn1Module ' "ASN1 DOCNAME" '
```

This action provides the complete information about an attribute in textual form. You can provide either the ASN.1 module name or the oid as the argument.

3.3.4 `getObjectClass` Action

```
getObjectClass ' "GDMO DOCNAME":objectClass '
```

This action provides the complete information about a class in textual form, which includes all the class attributes and their properties. You can provide either the class name or the oid as the argument.

3.3.5 `getDocument` Action

```
getDocument ' "GDMO DOCNAME" '
```

This action provides a list of all objects (and the oids) defined in a particular document. This action expects the document name.

3.3.6 `getPackage` Action

This action provides the following information about the content of the given package:

- Complete package name
- All package attributes

For each attribute, the `getPackage` action displays the following information:

 - Type information
 - Default value (if any)
 - Initial value (if any)
 - Permitted values (if any)
 - Required values (if any)
 - Properties (such as `GET`, `REPLACE`, `ADD`, `REMOVE`, and `REPLACE-WITH-DEFAULT`)
- All package actions

For each package action, the `getPackage` action displays the following information:

 - Action name
 - Information syntax of the action
 - Reply syntax of the action (if any)
- All package notifications

For each notification, the `getPackage` action displays a list of all attribute IDs. And for each attribute ID, the action displays the following information:

 - Notification name
 - Attribute full name
 - Field name
 - Label name

CODE EXAMPLE 3-1 shows the input syntax (information syntax) and the output syntax (reply syntax) of the `getPackage` action. The input syntax can be either the fully qualified name of the package (for example, `docLabel : { document "CD", label "cdPlayOptionsPackage" }`), or the object identifier of the package (for example, `oid : {1 3 6 1 4 1 42 2 2 3 20 1 6 1}`).

CODE EXAMPLE 3-1 Input/Output Syntax of the `getPackage` Action

```
Input Syntax is CHOICE {
    docLabel SEQUENCE {
        document [UNIVERSAL 19] IMPLICIT OCTET STRING (SIZE (0 ..
64)),
        label    [UNIVERSAL 19] IMPLICIT OCTET STRING (SIZE (0 ..
64))
    },
    oid          OBJECT IDENTIFIER
}
Result Syntax is SEQUENCE {
    oid          [0] IMPLICIT OBJECT IDENTIFIER,
    labels       SEQUENCE OF SEQUENCE {
        document [0] IMPLICIT OCTET STRING (SIZE (0 .. 64)),
        label    [1] IMPLICIT OCTET STRING (SIZE (0 .. 64))
    },
    attributes   SEQUENCE OF SEQUENCE {
        oid          [0] IMPLICIT OBJECT IDENTIFIER,
        labels       SEQUENCE OF SEQUENCE {
            document [0] IMPLICIT OCTET STRING (SIZE (0 .. 64)),
            label    [1] IMPLICIT OCTET STRING (SIZE (0 .. 64))
        },
        types        SEQUENCE {
            name ASN-1.Reference,
            type ASN-1.Asn1-Type
        },
        defaultval   SEQUENCE {
            value ASN-1.Asn1-Parsed-Value OPTIONAL
        },
        initialval   SEQUENCE {
            value ASN-1.Asn1-Parsed-Value OPTIONAL
        },
        permittedvals SEQUENCE {
            value ASN-1.Asn1-Type OPTIONAL
        },
        requiredvals SEQUENCE {
            value ASN-1.Asn1-Type OPTIONAL
        },
        props        SET OF ENUMERATED {
            get(0),
```

CODE EXAMPLE 3-1 Input/Output Syntax of the `getPackage` Action (*Continued*)

```

        replace(1),
        add(2),
        remove(3),
        replace-with-default(4)
    }
},
actions      SET OF SEQUENCE {
    document  [0] IMPLICIT OCTET STRING (SIZE (0 .. 64)),
    label     [1] IMPLICIT OCTET STRING (SIZE (0 .. 64)),
    infosyntax SEQUENCE {
        module PrintableString,
        name   PrintableString
    },
    replsyntax SEQUENCE {
        module PrintableString,
        name   PrintableString
    } OPTIONAL
},
notifications SET OF SEQUENCE {
    document  [UNIVERSAL 19] IMPLICIT OCTET STRING (SIZE (0
.. 64)),
    label     [UNIVERSAL 19] IMPLICIT OCTET STRING (SIZE (0
.. 64)),
    attributeids SET OF SEQUENCE {
        oid    [0] IMPLICIT OBJECT IDENTIFIER,
        field  [1] IMPLICIT OCTET STRING (SIZE (0 .. 64)),
        label  [2] IMPLICIT OCTET STRING (SIZE (0 .. 64))
    }
}
}

```

CODE EXAMPLE 3-2 shows the output of the `getPackage` action.

CODE EXAMPLE 3-2 Output of the `getPackage` Action Example

```

On running the "mdr_action" sample program
    from /opt/SUNWconn/em/src/pmi_hi

# ./mdr_action -a getPackage -l 'docLabel :
    { document "CD", label "cdPlayOptionsPackage" }'

Input Morf--> {
    document "CD",
    label "cdPlayOptionsPackage"
}

```

CODE EXAMPLE 3-2 Output of the `getPackage` Action Example (*Continued*)

```

Result --> {
    oid "CD":cdPlayOptionsPackage,
    labels {
        {
            document "CD",
            label "cdPlayOptionsPackage"
        }
    },
    attributes {
        {
            oid "CD":cdPlayList,
            labels {
                {
                    document "CD",
                    label "cdPlayList"
                }
            },
            types {
                name "TrackList",
                type choice : {
                    {
                        name "all",
                        type null : NULL
                    },
                    {
                        name "selection",
                        type set-of : defined : {
                            module "CD-asnlModule",
                            name "TrackId"
                        }
                    }
                }
            },
            defaultval {
                value defined : {
                    module "CD-asnlModule",
                    value "playAllTracks"
                }
            },
            initialval {
                value defined : {
                    module "CD-asnlModule",
                    value "playAllTracks"
                }
            },
            permittedvals {

```


CODE EXAMPLE 3-2 Output of the `getPackage` Action Example (*Continued*)

```

        requiredvals {
        },
        props {
            get,
            replace,
            add,
            remove,
            replace-with-default
        }
    },
    actions {
    {
        document "CD",
        label "cdPlayerPlayTrack",
        infosyntax {
            module "CD-asn1Module",
            name "TrackId"
        }
    }
    },
    notifications {
    {
        document "Rec. X.721 | ISO/IEC 10165-2 : 1992",
        label "attributeValueChange",
        attributeids {
            {
                oid "Rec. X.721 | ISO/IEC 10165-2 :
1992":sourceIndicator,
                field "sourceIndicator",
                label ""Rec. X.721 | ISO/IEC 10165-2 :
1992"":sourceIndicator"
            },
            {
                oid "Rec. X.721 | ISO/IEC 10165-2 :
1992":attributeIdentifierList,
                field "attributeIdentifierList",
                label ""Rec. X.721 | ISO/IEC 10165-2 :
1992"":attributeIdentifierList"
            },
            {
                oid "Rec. X.721 | ISO/IEC 10165-2 :
1992":attributeValueChangeDefinition,
                field "attributeValueChangeDefinition",
                label ""Rec. X.721 | ISO/IEC 10165-2 :
1992"":attributeValueChangeDefinition"
            },
        }
    }
    }

```


CODE EXAMPLE 3-3 shows the input syntax (information syntax) and the output syntax (reply syntax) of the `getPackagesByOC` action. The input syntax can be either the fully qualified name of the object class (for example, `docLabel : { document "Rec. X.721 | ISO/IEC 10165-2 : 1992", label "log" }`), or the object identifier of the object class (for example, `oid : {2 9 3 2 3 6}`).

CODE EXAMPLE 3-3 Input and Output Syntax for the `getPackagesByOC` Action Example

```
Input Syntax is CHOICE {
  docLabel SEQUENCE {
    document [UNIVERSAL 19] IMPLICIT OCTET STRING (SIZE (0 ..
64)),
    label    [UNIVERSAL 19] IMPLICIT OCTET STRING (SIZE (0 ..
64))
  },
  oid       OBJECT IDENTIFIER
}
Result Syntax is SET OF SEQUENCE {
  document [UNIVERSAL 19] IMPLICIT OCTET STRING (SIZE (0 .. 64)),
  label    [UNIVERSAL 19] IMPLICIT OCTET STRING (SIZE (0 .. 64)),
  type     ENUMERATED {
    mandatory(0),
    conditional(1)
  },
  hierarchy ENUMERATED {
    originated(0),
    inherited(1)
  } OPTIONAL
}
```

CODE EXAMPLE 3-4 shows the output of the `getPackagesByOC` action example.

CODE EXAMPLE 3-4 Output of the `getPackagesByOC` Action Example

```
On running the "mdr_action" sample program from
/opt/SUNWconn/em/src/pmi_hi

# ./mdr_action -a getPackagesByOC -l 'docLabel :
{ document "Rec. X.721 | ISO/IEC 10165-2 : 1992", label "log" }'

Input Morf--> {
  document "Rec. X.721 | ISO/IEC 10165-2 : 1992",
  label "log"
}
Result --> {
  {
    document "Rec. X.721 | ISO/IEC 10165-2 : 1992",
    label "logPackage",
```

CODE EXAMPLE 3-4 Output of the `getPackagesByOC` Action Example *(Continued)*

```

        type mandatory
    },
    {
        document "Rec. X.721 | ISO/IEC 10165-2 : 1992",
        label "finiteLogSizePackage",
        type conditional
    },
    {
        document "Rec. X.721 | ISO/IEC 10165-2 : 1992",
        label "logAlarmPackage",
        type conditional
    },
    {
        document "Rec. X.721 | ISO/IEC 10165-2 : 1992",
        label "availabilityStatusPackage",
        type conditional
    },
    {
        document "Rec. X.721 | ISO/IEC 10165-2 : 1992",
        label "duration",
        type conditional
    },
    {
        document "Rec. X.721 | ISO/IEC 10165-2 : 1992",
        label "dailyScheduling",
        type conditional
    },
    {
        document "Rec. X.721 | ISO/IEC 10165-2 : 1992",
        label "weeklyScheduling",
        type conditional
    },
    {
        document "Rec. X.721 | ISO/IEC 10165-2 : 1992",
        label "externalScheduler",
        type conditional
    }
}

```

3.3.8 `getOidName` Action

```
getOidName '{oid}'
```

For a given object identifier, returns the name of the object.

3.3.9 Sample MDR Action Program

CODE EXAMPLE 3-5 gives a sample program so you can try out these different actions of the MDR.

CODE EXAMPLE 3-5 MDR Actions

```
#include hi.hh
#include stdio.h

Platform plat;
main(int argc, char **argv)
{
    char dn[1024];
    if (argc != 4) {
        printf("Usage dummy: mdr hostname < MDR action>
<arglist> \n");
        printf("\nSupported Actions:\n \n");
        printf("\t getObjectClass \"GDMO
DOCNAME\":objectClass'\n");
        printf("\t getAllDocuments getAllDocuments\n");
        printf("\t getDocument \"GDMO DOCNAME\"'\n");
        printf("\t getAttribute \"GDMO
DOCNAME\":attrName'\n");
        printf("\t getAsn1Module \"ASN1 DOCNAME\"'\n");
        printf("\t getOidName '{oid}'\n");
        printf("\nSample Usage commands:\n\n");
        printf("\t mdr host getAsn1Module \"EM-TOPO-
ASN1\"'\n");
        printf("\t mdr host getOidName '{ 1 3 6 1 4 1 42
2 2 2 5 3 1
}'\n");
        printf("\t mdr host getAttribute \"EM
TOPOLOGY\":topoNodeName'\n");
        printf("\t mdr host getDocument \"EM
Topology\"'\n");
        printf("\t mdr host getObjectClass \"EM
TOPOLOGY\":topoNode'\n");
        printf("\t mdr host getAllDocuments
getAllDocuments\n");
        exit(0);
    }
    char *host = argv[1];
    char *host = argv[1];
    char *action = argv[2];
    char *mod = argv[3];
}
```

Symbolic Constants

CODE EXAMPLE 3-5 MDR Actions (Continued)

```
#include hi.hh
    plat = Platform(duEM);
    printf("Connecting to %s ... ",host);
    plat.connect(host, "test_get");
    printf("Done.\n");
    printf(dn,"/systemId=\"%s\"/metaName=\"MDR\"", host);
    Image mdr = Image(dn);
    mdr.boot();
    Syntax in = mdr.get_param_syntax(action);
    Syntax res = mdr.get_result_syntax(action);
    printf("Input Syntax is %s\n",in.get().chp());
    printf("\n-----\n");
    printf("Result Syntax is %s\n",res.get().chp());
    DU mdr_data = mdr.call(action, mod);
    printf("--> %s\n",mdr_data.chp());
}
```

3.4 Symbolic Constants

The following is an example of a symbolic constant.

```
const Timeout DEFAULT_TIMEOUT = -12345.0;
```

`DEFAULT_TIMEOUT` is the default argument to several of the member functions of the `Platform` class. `-12345.0` is a distinguished value that causes those functions to substitute the value of the `TIME_OUT` property from the `Platform` instance. Refer to Section 3.5.5.2 “Timeout” on page 3-24,” for more information. For example:

```
const DU duREPLACE = "REPLACE";

const Callback NO_CALLBACK;

enum Platformid
{
    VOID_PLATFORM_ID,
    G2_PLATFORM_ID,
};
```

Symbolic Constants

TABLE 3-3 is a list of often used string constants.

TABLE 3-3 String Constants

Constant	Definition
duACCESS	"ACCESS"
duACCESS_DENIED	"ACCESS_DENIED"
duAPPLICATION_OBJNAME	"APPLICATION_OBJNAME"
duAPPLICATION_TYPE	"APPLICATION_TYPE"
duATTR_CHANGED	"ATTR_CHANGED"
duAUTOIMAGE	"AUTOIMAGE"
duBOOT	"BOOT"
duCHANGED	"CHANGED"
duDEFAULT	"DEFAULT"
duDEFAULT_ALLOWED	"DEFAULT_ALLOWED"
duDEFAULT_TIMEOUT	"DEFAULT_TIMEOUT"
duDERIVATION	"DERIVATION"
duDISCONNECTED	"DISCONNECTED"
duDOWN	"DOWN"
duEFFECTIVE_USER	"EFFECTIVE_USER"
duERROR	"PMI_ERROR"
duEXCLUDE	"EXCLUDE"
duEXCLUDE_ALLOWED	"EXCLUDE_ALLOWED"
duEXISTS	"EXISTS"
duFALSE	"FALSE"
duIGNORE	"IGNORE"
duIGNORE_ALLOWED	"IGNORE_ALLOWED"
duIMAGE_EXCLUDED	"IMAGE_EXCLUDED"
duINCLUDE	"INCLUDE"
duINCLUDE_ALLOWED	"INCLUDE_ALLOWED"
duLAST_ERROR	"LAST_ERROR"
duLOCATION	"LOCATION"
duMAYBE	"MAYBE"
duMISC_EVENT	"MISC_EVENT"

Symbolic Constants

TABLE 3-3 String Constants *(Continued)*

Constant	Definition
duMODIFIABLE	"MODIFIABLE"
duMOD_PENDING	"MOD_PENDING"
duNICKNAME	"NICKNAME"
duNICKNAME_IS_PERMANENT	"NICKNAME_IS_PERMANENT"
duNO	"NO"
duNONE	"NONE"
duNOT_LOADED	"NOT_LOADED"
duNO_VALUE	"NO_VALUE"
duOBJCLASS	"OBJCLASS"
duOBJECT_CREATED	"OBJECT_CREATED"
duOBJECT_DESTROYED	"OBJECT_DESTROYED"
duOBJNAME	"OBJNAME"
duOBJFULLNAME	"OBJFULLNAME"
duOWNERSHIP	"OWNERSHIP"
duEM	"EM"
duPLATFORM_NICKNAME	"PLATFORM_NICKNAME"
duPLATFORM_OBJNAME	"PLATFORM_OBJNAME"
duPLATFORM_TYPE	"PLATFORM_TYPE"
duPOLL	"POLL"
duRAW_EVENT	"RAW_EVENT"
duREPLACE	"REPLACE"
duREPLACE_ALLOWED	"REPLACE_ALLOWED"
duSHARED	"SHARED"
duSHUTDOWN	"SHUTDOWN"
duSNAP	"SNAP"
duSTATE	"STATE"
duTICKET	"TICKET"
duTRACK	"TRACK"
duTRACKMODE	"TRACKMODE"
duTRUE	"TRUE"

Defined Types

TABLE 3-3 String Constants *(Continued)*

Constant	Definition
duUP	"UP"
duUSER	"USER"
duWAIT	"WAIT"
duYES	"YES"
duACTUALCLASS	"ACTUAL_CLASS"

3.5 Defined Types

The defined types are shown in this section. They are declared in the `/opt/SUNWconn/em/include/pmi/hi.hh` file.

3.5.1 Asn1Int

```
typedef GenInt Asn1Int;
```

3.5.2 CCB

```
typedef const Callback& CCB;
```

3.5.3 CDU

```
typedef const DU& CDU;
```

3.5.4 DU

```
typedef DataUnit DU;
```

3.5.5 FBits

```
typedef U32 FBits;
```

The bits have the meanings on a get shown in TABLE 3-4.

TABLE 3-4 Format Bit Values on get Function Calls

Format Bit	Description
USE_NUMERIC_NAMES	Do not translate OIDs to names
OMIT_NEWLINES	Do not “pretty-print”. Format only one line of output
USE_C_ESCAPES	Format control characters as C does: \n, \033, etc
USE_EXPLICIT_TYPES	Format type tags on ANY values

Defined Types

TABLE 3-4 Format Bit Values on `get` Function Calls (*Continued*)

Format Bit	Description
OMIT_SPACES	Omit all nonessential space characters
USE_HEX	Formats the string in the usual C hex format with a leading 0x. Valid for octet strings only.
USE_EXPLICIT_CHOICE	Format choice with an explicit choice tag

The bits have the meanings on a `set` as shown in TABLE 3-5.

TABLE 3-5 Format Bit Values on `set` Function Calls

Format Bit	Description
USE_NUMERIC_NAMES	(Ignored)
OMIT_NEWLINES	(Ignored)
USE_C_ESCAPES	Parse control characters as C does: <code>\n</code> , <code>\033</code> , etc.
USE_EXPLICIT_TYPES	Require type tags on ANY values
OMIT_SPACES	(Ignored)
USE_HEX	(Ignored)
USE_EXPLICIT_CHOICE	Expect choice to have an explicit choice tag.

3.5.5.1 FormatBits

```
Enum FormatBits
{
    USE_NUMERIC_NAMES    = 1,
    OMIT_NEWLINES        = 2,
    USE_C_ESCAPES         = 4,
    USE_EXPLICIT_TYPES    = 8,
    OMIT_SPACES           = 16,
    USE_HEX               = 32,
    USE_EXPLICIT_CHOICE    = 65536
};
```

3.5.5.2 Timeout

```
typedef double Timeout ;
```

3.6 Error Handling and Event Dispatching

Error handling is provided by the base class `Error`. Each of the object classes is derived from the `Error` class, except for the class, `AlbumImage`.

The function `dispatch_recursive()` maintains queues of callback routines. One queue is maintained for each of the following: input, output, exception, and timers. These queues are scanned in the following order:

- Exception
- Output
- Timer
- Input

You associate a file descriptor with each callback on a queue when you use a function such as `post_fd_read_callback()`. When you call `dispatch_recursive()`, this function does a select on all the open file descriptors to determine their state, and then goes through each queue in the order indicated above to determine if there is outstanding data to be read from, or written to, the file descriptor.

pmi_sched_get_fds Function

Either `FALSE` or `TRUE` can be passed as a value to the dispatcher as an argument. If `FALSE` is the value passed, then the select on the open file descriptors is done with a time-out value of 0. If `TRUE` is passed, then a short time interval is specified as the time-out.

3.6.1 Event Dispatching Functions

The following functions maintain the queues of the callback routines.

```
void dispatch_main_loop(); // Event dispatch queue for user applications
```

```
void dispatch_recursive(Boolean wait); // Flush scheduler buffer function
```

The following functions maintain critical region handling.

```
int writecallback_exists(); // Checks callback queue for pending writes
```

```
void flush_events_callbacks(); // Flush all pending events to the MIS
```

3.7 pmi_sched_get_fds Function

pmi_sched_get_fds

```
void pmi_sched_get_fds(fd_set &rd_mask, fd_set &wr_mask, fd_set  
&expt_mask)
```

The `pmi_sched_get_fds` function returns the file descriptors that are set by the scheduler for read, write and exception conditions into the corresponding file descriptors `rd_mask`, `wr_mask`, `expt_mask`.

3.8 High-Level PMI Classes

Each of the classes shown in TABLE 3-6 is implemented as a reference-counting outer class wrapped around an inner abstract-base class. The PMI requires no access to the inner class.

TABLE 3-6 High-Level PMI Classes

Class	Description
Album Class	Represents a set of related objects
AlbumImage Class	Represents the state of an iterator
AppTarget Class	Represents target applications
AuthApps Class	Used when you need to know which applications a user is authorized to use
AuthFeatures Class	Used to implement the feature level access control in your application
Coder Class	Represents a pair of methods for encoding and decoding values
CurrentEvent Class	Represents an event
Error Class	Stores details of errors related to an object instance
Image Class	Represents an actual or potential object in an MIS' framework
Morf Class	Represents a unit of data
MorfBuilder Class	Provides flexibility with Morf objects
PasswordTty Class	Implements the TTY based password query mechanism
Platform Class	Represents a potential or actual connect to a MIS
Syntax Class	Represents a type
Waiter Class	Represents an ongoing asynchronous operation

Note – As all class destructors have an identical format, such as `~class name()`, the various class destructors are not specified in the following sections. Each class destructor might or might not destroy the underlying *class name* object, depending on the reference count.

3.9 Album Class

Inheritance: public Error

```
#include pmi/hi.hh
```

Data Members: No public data members are declared in this class.

An album contains a set of related objects, implemented as a set of `images`. Like mathematical sets, albums can be constructed either by rule or by enumeration. An album instance allows you to perform certain operations on each of the `images` that it contains. Like `images`, albums can be synchronized either manually or automatically.

The Album class allows an attribute list to be specified as an argument; in this way memory consumption of `images` is reduced. As examples, see `derive()` and `start_derive()`.

TABLE 3-7 Album Method Types

Method Name	Method Type
<code>find_by_nickname</code>	Global lookup
<code>get_prop</code> <code>set_prop</code> <code>all_set_prop</code> <code>all_set_attr_prop</code>	Control properties
<code>get_derivation</code> <code>set_derivation</code> <code>derive</code> <code>start_derive</code>	Define membership of a derived album
<code>include</code> <code>exclude</code> <code>clear</code>	Manipulate membership of an enumerated album
<code>num_images</code>	Census info
<code>get_userdata</code> <code>set_userdata</code>	User-defined properties
<code>first_image</code> <code>all</code>	Iterate over all images

TABLE 3-7 Album Method Types *(Continued)*

Method Name	Method Type
all_boot all_start_boot all_shutdown all_start_shutdown	Image activation
start_m_get start_m_set start_m_action start_m_action_raw start_m_delete	asynchronous CMIS
all_set all_set_long all_set_str all_set_gint all_set_dbl all_set_raw all_revert all_store all_start_store all_set_from_ref	Setting attributes
all_create all_start_create all_create_within all_start_create_within all_destroy all_start_destroy	Object existence
all_call all_start_raw all_start	Miscellaneous methods
all_when	Object events
get_when_syntax when	Album events

Note – When in use, the `start_m_get()`, `start_m_set()`, `start_m_action()`, `start_m_action_raw()`, and `start_m_delete()` methods do *not* populate the album like the `start_derive()` method does. Only the `start_derive()` and `include()` methods populate the album with images. Also, the existing `all_start()` method and the new `all_start_raw()` method only address already populated albums. For more information, refer to the examples of `start_m_get()` and `all_start()` method utilization.

3.9.1 Constructors

Default Constructor

```
Album()
```

The default constructor creates an album instance that refers to no actual album object. The value tests `FALSE` until you assign it a real album value.

Copy Constructor

```
Album(const Album& other);
```

This is an ordinary copy constructor. After the copy, both copies still refer to the same album object. The reference count on the album object is incremented.

Album *Constructor*

```
Album(CDU nickname,  
      Platform& p = Platform::default_platform(), )
```

This constructor constructs an album instance for a particular kind of `Platform`. Because an album is really a wrapper for a set of related classes, this function actually works somewhat like a virtual constructor.

Note – A *nickname* is an album's unique identifier. If you create two albums with the same nickname in one application, the second will be the same as the first.

3.9.2 Album Operator Overloading

Assignment Operator

```
Album& operator = (const Album& other)
```

The assignment operator works the same as the copy constructor.

Cast Operator

```
operator void *();
```

The cast operator is for use in conditionals. It returns TRUE if this album refers to an actual album object. Do not attempt to use the returned value as a pointer to anything, since it points to private data.

Not Operator

```
operator !();
```

This is provided so that you can say “if (!album) ...”

3.9.3 Album Member Functions

This section describes the member functions of the Album class.

`all`

Purpose: Apply a function, provided as an argument, over all of the images in the album.

```
virtual Result all(Result (*f)( Image &im, void* data ),  
                  void* data = 0)
```

Album Class

- The first argument *f* is a pointer to the subfunction.
- The next argument *im* is the instance of the image object class, supplied automatically.
- The next argument *data* is some arbitrary data to pass to the subfunction along with each image.

The subfunction's syntax (embedded in the declaration of `all`) is therefore required to be:

```
Result (*f)(Image &im, void* data) ;
```

The implementation of `all` calls the subfunction repeatedly, supplying an appropriate value for *im* to refer in turn to each of the album's images, and passing to it each time the arbitrary data that was passed to `all`.

The `all` function itself returns a `TRUE` value if all the calls to the subfunction designated by *f* return `TRUE`. A thrown exception terminates the iteration.

`all_boot`

Purpose: Perform a `boot` on each of the images in the album.

```
Result all_boot(Timeout to = DEFAULT_TIMEOUT)
```

The timeout value is reset on completion of each successful boot. This function returns a `TRUE` value if all image operations succeeded.

Example: Boot all images in an album.

```
Album bunch = Album("demoalbum"); // Define, construct bunch.
bunch.set.derivation( "/LV(2)" ); // Derive the album.
Timeout to ;
...
if ( !bunch.all_boot( to ) ) {           // Boot all images in bunch.
    cout << "Using all_boot(): boot of one image failed." ;
    exit ( 1 ) ;
}
```

Album Class

all_call

Purpose: Perform a `call` on each of the images in the album.

```
Result all_call(CDU name,
               CDU param,
               const Timeout to = DEFAULT_TIMEOUT)
```

name is the name of the action.

param is the parameter associated with the action.

to is the timeout.

If *param* is not provided (or if it is set to `duNONE`), there is no parameter associated with this action. The time-out is reset on completion of each successful call. Returns a `TRUE` value if all image operations succeeded.

Example: Call all images in an album.

```
Album bunch = Album("demoalbum"); // Define, construct bunch.
bunch.set.derivation( "/LV(2)" ); // Derive the album.
CDU nm, prm ;
Timeout to ;
...
if (!bunch.all_call( nm, prm, to)) { // Call all images in bunch.
    cout << "Using all_call(): call of one image failed." ;
    exit ( 1 ) ;
}
```

all_create

Purpose: Perform a `create` on each image in the album.

```
Result all_create( Image &refobj = Image(),
                  const Timeout to = DEFAULT_TIMEOUT)
```

The `Timeout` value is reset on completion of each successful `create` call. Returns a `TRUE` value if all image operations succeeded.

Album Class

Example: Create an object for each image in an album.

```
Album bunch = Album("demoalbum"); // Define, construct bunch.
bunch.set.derivation( "LV(2)" ); // Derive the album.
Image robj;
Timeout to ;
// Set name and class before invoking bunch.all_create().See
create.cc
// Create object of each image in bunch.
if ( !bunch.all_create(robj, to)) {
    cout << "Using all_create(): create of one object failed." ;
    exit ( 1 ) ;
}
```

all_create_within

Purpose: Perform a create_within on each of the images in the album.

```
Result all_create_within(CDU container_objname,
    Image &refobj = Image(),
    Timeout to = DEFAULT_TIMEOUT)
```

The Timeout value is reset on completion of each successful create_within call. Returns a TRUE value if all image operations succeeded.

Example: Create an object within a container, for each image in an album.

```
// Create object of each image in bunch.
if ( !bunch.all_create_within(cobj, robj, to)) {
    cout << "Using all_create_within(): create of one object failed.";
    exit ( 1 ) ;
}
```

all_destroy

Purpose: Perform a destroy on each of the images in the album.

```
Result all_destroy(Timeout to = DEFAULT_TIMEOUT)
```

The Timeout value is reset on completion of each successful destroy call. Returns a TRUE value if all image operations succeeded.

Album Class

Example: Destroy each image in an album.

```
if ( !bunch.all_destroy(to)) { // Create object of each image in
bunch.
    cout << "Using all_destroy(): create of one object failed." ;
    exit ( 1 ) ;
}
```

all_revert

Purpose: Perform a revert on each of the images in the album.

```
Result all_revert()
```

Returns a TRUE value if all image operations succeeded.

all_set

Purpose: Perform a set on each of the images in the album.

```
Result all_set(CDU name,
               CDU value,
               CDU op = duREPLACE)
```

Returns a TRUE value if all image operations succeeded.

all_set_attr_prop

Purpose: Set a property of an attribute in each of the images of current MIS.

```
Result all_set_attr_prop(CDU name,
                          CDU key,
                          CDU value)
```

This function sets a property for some attribute in each of the images of the current MIS, provided that:

- *name* specifies an existing attribute in the object class
- *key* specifies a supported property

Album Class

- *value* specifies a legal value.

If `all_set_attr_prop` cannot do what you ask, it throws an invalid exception.

Refer to `get_attr_prop`, under the description of `Image`, for some typical properties. Returns a `TRUE` value if all image operations succeeded.

`all_set_dbl`

Purpose: Perform a `set_dbl` on each of the `images` in the album.

```
Result all_set_dbl(CDU name,  
                  double value,  
                  CDU op = duREPLACE)
```

Returns a `TRUE` value if all image operations succeeded.

`all_set_from_ref`

Purpose: Perform a `set_from_ref` on each of the `images` in the album.

```
Result all_set_from_ref( Image& refobj)
```

Returns a `TRUE` value if all image operations succeeded.

`all_set_gint`

Purpose: Perform a `set_gint` on each of the `images` in the album.

```
Result all_set_gint(CDU name,  
                   GenInt& value,  
                   CDU op = duREPLACE)
```

Returns a `TRUE` value if all image operations succeeded.

`all_set_long`

Purpose: Perform a `set_long` on each of the

Album Class

s in the album.

```
Result all_set_long(CDU name,
                   long value,
                   CDU op = duREPLACE)
```

Returns a TRUE value if all image operations succeeded.

all_set_prop

Purpose: Set a property for each of the images of the current album.

This function sets a property for each of the images of the current album, provided that:

- *key* specifies a supported property.
- *value* specifies a legal value.

If all_set_prop cannot do what you ask, it throws an invalid exception.

```
Result all_set_prop(CDU key, CDU value)
```

Refer to get_prop, under the description of Image, for some typical properties. Returns a TRUE value if all image operations succeeded.

all_set_raw

Purpose: Perform a set_raw on each of the images in the album. Returns a TRUE value if all image operations succeeded.

```
Result all_set_raw(CDU name,
                  Morf& value,
                  CDU op = duREPLACE)
```

all_set_str

Purpose: Perform a set_str on each of the images in the album.

Returns a TRUE value if all image operations succeeded.

```
Result all_set_str(CDU name,
                  CDU value,
                  CDU op = duREPLACE,
                  FBits fb = 0)
```

The difference between `set_str` and `set` is that the data language used by `set` requires quotes as part of the string, while `set_str` assumes them if necessary. (They are not always necessary; you can also pass numeric values as strings, and they are converted for you.)

Refer to TABLE 3-13 for a description of legal operations. For a list of possible *fb* values, refer to Section 3.5.5.1 “FormatBits” on page 3-24.”

`all_shutdown`

Purpose: Perform a shutdown on each of the images in the album.

The Timeout value is reset on completion of each successful shutdown. Returns a TRUE value if all image shutdowns succeed.

```
Result all_shutdown(const Timeout to = DEFAULT_TIMEOUT)
```

`all_start`

Purpose: Perform an asynchronous version of `all_call`.

The callback is called only once when all images are complete.

```
Waiter all_start(CDU name,
                 CDU param,
                 CCB cb = NO_CALLBACK)
```

`all_start_boot`

Purpose: Perform an asynchronous version of `all_boot`.

Album Class

The callback is called only once, when all images are complete.

```
Waiter all_start_boot(CCB cb = NO_CALLBACK)
```

`all_start_create`

Purpose: Perform an asynchronous version of `all_create`.

The callback is called only once, when all images are complete.

```
Waiter all_start_create( Image& refobj = Image(),  
                        CCB cb = NO_CALLBACK)
```

`all_start_create_within`

```
Waiter all_start_create_within(CDU container_objname,  
                               Image & refobj = Image(),  
                               CCB cb = NO_CALLBACK)
```

The preceding function call is the asynchronous version of `all_create_within`. The callback is called only once, when all images are complete.

`all_start_destroy`

```
Waiter all_start_destroy (CCB cb = NO_CALLBACK)
```

The preceding function call is the asynchronous version of `all_destroy`. The callback is called only once, after all images are destroyed.

Album Class

`all_start_raw`

```
Waiter all_start_raw(CDU name, Morf param,  
                    CCB cb = NO_CALLBACK)
```

Performs a `start_raw()` action asynchronously on each image in the album. This method offers exactly the same functionality as the existing `Waiter all_start` method. It differs in that the parameter *param* is a Morf rather than a DataUnit.

The callback is called only once when all asynchronous `start_raw()` or `start()` actions on images are completed.

Note – The `all_start` method performs a `start()` asynchronously on each image in an album.

`all_start_shutdown`

```
Waiter all_start_shutdown(CCB cb = NO_CALLBACK)
```

The preceding function call is the asynchronous version of `all_shutdown`. The callback is called only once, after all image shutdowns are complete.

`all_start_store`

```
Waiter all_start_store(CCB cb = NO_CALLBACK)
```

The preceding function call is the asynchronous version of `all_store`. The callback is called only once, after all image stores are complete.

`all_store`

```
Result all_store(const Timeout to = DEFAULT_TIMEOUT)
```

The preceding function call performs a `store` on each of the images in the album. The Timeout value is reset on completion of each successful store. It returns a TRUE value if all image stores succeed.

Album Class

all_when

```
Result all_when(CDU eventname,  
                CCB cb = NO_CALLBACK)
```

The preceding function call performs a `when` on each of the images in the album. For a list of supported events, refer to `when`, under the description of `Image`. Returns a `TRUE` value if all image when operations succeed.

clear

```
Result clear()
```

The preceding function call nulls out this album, which is presumably of the enumerated variety. (The `clear` function is not as useful with derived albums, since such albums either track membership automatically or are repopulated by calling `Album::derive` again.)

derive

```
Result derive(const Timeout to = DEFAULT_TIMEOUT)
```

The preceding function call causes the album membership list to be computed (or recomputed) using the derivation specified in the property `DERIVATION`. All previous membership information is lost. You can initialize a non-tracking album with a `derive` and then maintain it with `include` and `exclude`. A tracking album does not track until the first `derive` is done, so it works much like a `boot` does on an image.

derive

```
virtual Result derive (Array (DU) attrlist,  
                      Timeout to = DEFAULT_TIMEOUT)= 0;
```

This function call allows a user to supply a list of attributes to be tracked when deriving an auto-imaging album. Attributes that are specified in the list are automatically tracked in each of the images in the derived album.

Album Class

exclude

```
Result exclude(Album& *ai)
```

This function call deletes from this album the set of images that are in the album *ai*.

Note – Inclusion and exclusion are simple procedural operations for use on enumerated albums, and do not “define” the membership of the album in any way that would override subsequent membership operations.

find_by_nickname

```
static Album find_by_nickname(CDU name,  
                             Platform& plat = Platform::def_platform())
```

The preceding function call looks up an album by its nickname and returns a pointer to the corresponding instance of album. Because albums do not generally have object names, this is the only way to find an album by name.

first_image

```
AlbumImage first_image()
```

The preceding function call returns the first image in the album, in the form of an AlbumImage iterator. The AlbumImage::next_image function returns subsequent images until there are no more.

Note – If you are tempted to use this iterator to see whether an image is a member of an album, remember that the Image::is_in_album function does this much more efficiently.

Album Class

get_derivation

DU get_derivation()

The preceding function call is a shortcut function for the `get_prop` function.

get_prop

Du get_prop(CDU *key*)

The preceding function call returns a property of the current album. If the *key* does not specify an existing property, a null `DataUnit` is returned, which tests `FALSE` in a conditional.

Most albums support the properties described in TABLE 3-8.

TABLE 3-8 Properties Supported by Most Albums

Properties	Description
STATE	Whether the album is actively tracking: DOWN - The album is not tracking. BOOT - The album is currently being derived. UP - The album is tracking. SHUTDOWN - The album is shutting down.
DERIVATION	How this album relates to other albums and images by scoping and filtering. Refer to Section 3.2.6 “Filtering as an Aspect of Album Derivation” on page 3-5.”
NICKNAME	The nickname of the album.
TRACKMODE	How the membership of the album is to be maintained: SNAP - the album does not maintain its membership list, but relies on explicit <code>derive()</code> , <code>include()</code> , and <code>exclude()</code> calls to tell the album which images are to be included. Note that a SNAP album might contain tracking images. TRACK — the album analyzes its derivation rule and automatically includes or excludes images as they change, when they match or fail to match the derivation rule. Refer to the <code>AUTOIMAGE</code> property if you want included images to boot and start tracking automatically.

TABLE 3-8 Properties Supported by Most Albums (*Continued*)

Properties	Description
AUTOIMAGE (YES, NO)	For any image included in the album, also automatically boots the image as a tracking image. (This happens before any of the registered callbacks, if any, are called.)
ACCESS (RWRWRW)	The access permissions for this album, if persistent.
BEST_EFFORT (YES, NO)	When set to 'NO', any error will return an error response message.

get_userdata

```
DU get_userdata(CDU key)
```

The preceding function call returns any data stored by the application under the *key* specified. There are no predefined values. If there is no data under that *key* for this instance, the return value (a null `DataUnit`) evaluates to `FALSE`.

get_when_syntax

```
Syntax get_when_syntax(CDU eventname)
```

The preceding function call returns the `Syntax` of a given event type. The event information comes into the callback as a `CurrentEvent`, from which the event information can be extracted.

include

```
Result include(Image& im)
```

The preceding function call adds the specified image to this album.

```
Result include(Album& al)
```

The preceding function call adds the set of images in the album *al* to this album.

Note – Inclusion and exclusion are simple procedural operations for use on enumerated albums, and do not “define” the membership of the album in any way that would override subsequent membership operations.

num_images

```
U32 num_images()
```

The preceding function call returns the number of `images` in this album.

set_derivation

```
Result set_derivation(CDU derivation)
```

The preceding function call is a shortcut function for the following:

```
Result set_prop(CDU key, CDU value)
```

The preceding function call sets a property of the current album, provided that *key* specifies a supported property and *value* specifies a legal value. If `set_prop` cannot do what you ask, it throws an invalid exception. Refer to `get_prop` under the description of the Album class for some typical properties.

set_userdata

```
Result set_userdata(CDU key, CDU value)
```

The preceding function call stores arbitrary data supplied by the application under the *key* specified. There are no predefined values. If this instance already has data under that *key*, the new data replaces the old without comment. Essentially, user data is an associative array belonging to the album; an application can use it in any way.

Album Class

start_derive

```
Waiter start_derive(CCB cb = NO_CALLBACK)
```

The preceding function call is the asynchronous version of `derive`.

```
virtual Waiter start_derive( Array (DU) attrlist,  
CCB cb = NO_CALLBACK)=0 ;
```

This function call is the asynchronous version of the `derive` function that accepts a list of attributes.

start_m_get

```
Waiter start_m_get(Array(DU) attrlist, CCB cb = NO_CALLBACK)
```

The preceding function performs an asynchronous CMIS `M_GET`. The scope and filter are built from the derivation string of the album. *attrlist* is the attribute list whose values need to be retrieved. The callback is called only once upon completion of the request.

start_m_set

```
Waiter start_m_set(Queue(AttrModifier)& attrmodq,  
CCB cb = NO_CALLBACK,CDU sync = duBEST_EFFORT)
```

The preceding function performs an asynchronous CMIS `M_SET`. The scope and filter are built from the derivation string of the Album. The parameter *attrmodq* represents a set of attributes, operations on the attributes, and, depending on the operation, the new values of these attributes.

The callback is called only once upon completion of the request. The parameter *sync* is intended for further implementation to indicate whether the synchronization mode is best effort or atomic. Currently, the supported mode is best effort.

start_m_action

```
Waiter start_m_action(CDU name, CDU param = duNONE,  
                     CCB cb = NO_CALLBACK, CDU sync = duBEST_EFFORT)
```

The preceding function performs an asynchronous CMIS `M_ACTION`. The scope and filter are built from the derivation string of the album. The *name* is the action name and *param* is the parameter associated with the action. The *param* is required in DataUnit format.

The callback is called only once upon completion of the request. The parameter *sync* is intended for further implementation to indicate whether the synchronization mode is best effort or atomic. Currently, the supported mode is best effort.

start_m_action_raw

```
Waiter start_m_action_raw(CDU name, Morf param = Morf(),  
                          CCB cb = NO_CALLBACK, CDU sync = duBEST_EFFORT)
```

The preceding function performs an asynchronous CMIS `M_ACTION`. The scope and filter are built from the derivation string of the album. The *name* is the action name and *param* is the parameter associated with the action. The *param* is required in Morf format.

The callback is called only once upon completion of the request. The parameter *sync* is intended for further implementation to indicate whether the synchronization mode is best effort or atomic. Currently, the supported mode is best effort.

start_m_delete

```
Waiter start_m_delete(CCB cb = NO_CALLBACK)
```

The preceding function performs an asynchronous CMIS `M_DELETE`. The scope and filter are built from the derivation string of the album. The callback is called only once upon completion of the request.

*Example***CODE EXAMPLE 3-6** start_m_get() and all_start() Method Utilization

```

#include netdb.h
#include sys/systeminfo.h

#include stdio.h
#include hi.hh
#include message.hh

void done_cb( Ptr userData, Ptr );
void asyn_cb( Ptr , Ptr calldata);

int main( int argc, char **argv)
{
    // Get the host name from the environment variable.

    char *host = getenv("EM_SERVER");
    if (!host) {
        host = new char[MAXHOSTNAMELEN+1];
        sysinfo(SI_HOSTNAME, host, MAXHOSTNAMELEN);
    }

    // Set the fdn for the reference object.
    char fdn[1024];
    sprintf(fdn, "/systemId=name:\"%s\"/topoNodeDBId=NULL/
LV(0)", host);

    Platform plat = Platform(duEM);
    if (plat.get_error_type() != PMI_SUCCESS) {
        cout << plat.get_error_string() << endl;
    exit(1);
    }

    cout << "Connecting to ... " << host << endl;

    // Connect to the host MIS.

    if (!plat.connect(host, "em_sample")) {
        cout << "Failed to connect to " << host << endl;
        cout << plat.get_error_string() << endl;
        exit(2);
    } else {
        cout << "Connected. " << endl;
    }

    // Construct test_album.

```

CODE EXAMPLE 3-6 start_m_get() and all_start() Method Utilization (*Continued*)

```

        Album test_album = Album("myalbum");
        if (test_album.get_error_type() != PMI_SUCCESS) {
            cout << test_album.get_error_string() << endl;
            exit(3);
        }

        // fill up the derivation property
        if (!test_album.set_derivation(fdn)) {
            cout << test_album.get_error_string() << endl;
            exit(4);
        }

        // fill up all kinds of properties on this album
        if (!test_album.set_prop(duREPORT_ANY_ERROR,duYES)) {
            cout << test_album.get_error_string() << endl;
            exit(5);
        }

        if (!test_album.set_prop(duREPORT_GET_LIST_ERROR,duYES)) {
            cout << test_album.get_error_string() << endl;
            exit(6);
        }

        if (!test_album.set_prop(duAUTOIMAGE,duYES)) {
            cout << test_album.get_error_string() << endl;
            exit(7);
        }

        // populate the album with image
        if (!test_album.derive()) {
            cout << test_album.get_error_string() << endl;
            exit(8);
        }

        cout << "The album contents " << test_album.num_images() << "
image(s)\n" << endl;

        // Will be set to 1 when Callback is done.
        int done = 0;

        Waiter cur;

        if
        (!(cur=test_album.all_start(DU("topoGetNodeReport"),DU("NULL"),C
allback(done_cb, &done)))) {
            cout << test_album.get_error_string() << endl;
            exit(9);
        }

```

CODE EXAMPLE 3-6 start_m_get() and all_start() Method Utilization (*Continued*)

```

    }
    if (cur.get_except()) {
        cout << cur.get_except()->reason() << endl;
        exit(10);
    }
    // subscribe to any future incoming replies
    cur.when_resp(Callback(asyn_cb,0));

    // Enter the listen loop to wait for asyn operation to
complete
    cout << "Waiting for asyn operation to complete...\n";
    cout << endl << endl;
    while (!done) {
        dispatch_recursive(TRUE);
    }

    exit(0);
}

void done_cb( Ptr userData, Ptr )
{
    cout << "\nExecuting THE LAST asyn callback function..."
<< endl;
    cout << "-----" <<
endl;
    cout << "After the operation is completed ";
    cout << "\n" << endl;

    // Set main done.
    (*(int*)userData)++;
}

void asyn_cb( Ptr , Ptr calldata)
{
    static int num = 1;
    cout << "\nExecuting asyn1 callback function for ";
    cout << num << " times";
    cout << endl;
    cout << "-----" <<
endl;
    num++;
    // Get and print the new attribute value.
    cout << "During the all_start operation ";
    cout << endl;
    if(calldata)
    {

```

CODE EXAMPLE 3-6 start_m_get() and all_start() Method Utilization (*Continued*)

```

        CurrentEvent ce(calldata);
        cout << "OBJNAME = " << ce.get_objname().chp() << endl;
        cout << "OBJCLASS = " << ce.get_objclass().chp() << endl;
        MessagePtr msg = (MessagePtr)ce.get_message();

        if(msg->type()==ACTION_RES)
        {
            ActionRes* srmsg = (ActionRes*)msg;
            cout << "OBJCLASS = " << oc2name(srmsg->oc).chp() << endl;
            cout << "FDN = " << oi2fdn(srmsg->oi).chp() << endl;
            cout << "ACTION-TYPE = " << endl;
            (srmsg->action_type).print(stdout);
            cout << "\n" << endl;
            cout << "ACTION-REPLY = " << endl;
            (srmsg->action_reply).print(stdout);
            cout << "\n" << endl;
        }

        Morf mf = ce.get_info_raw();
        AsnlValue val = mf.get_value();
        if(val)
        {
            cout << "info_raw() field of current event ACTION-REPLY = "
            << endl;
            val.print(stdout);
            cout << "\n" << endl;
        }
        cout << "eventtype() field of current event ACTION-TYPE = " <<
        ce.get_eventtype().chp() << endl;

        cout << "Information setting in the current event related
        Album " << endl;
        cout << "Derivation rule for the Album " <<
        ce.get_album().get_prop(duNICKNAME).chp() << " is : " <<
        ce.get_album().get_derivation().chp() << endl;
        cout << "\n" << endl;

        cout << "Information setting in the current event related
        Image" << endl;
        Image im = ce.get_image();
        cout << " image name is " << im.get_objname().chp() << endl;
        cout << " image class is " << im.get_objclass().chp() <<
        endl;
        cout << " image state is " << im.get_state().chp() << endl;
        cout << " image last_error is " << im.get_last_error().chp()
        << endl;
        if (im.exists())

```

CODE EXAMPLE 3-6 start_m_get() and all_start() Method Utilization (*Continued*)

```

        cout << " image exists " << endl;
    else
        cout << " image does not exist " << endl;
    cout << " attribute(s) and attribute value(s) setting in the
image " << endl;
    Array(DU) attr_names = im.get_attr_names();
    for (int i=0; i<attr_names.size; i++) {
        char *name = strdup(attr_names[i].chp());
        cout << name;
        cout << ": ";
        cout << im.get_str(name).chp() << endl;
    }
    cout << "\n" << endl;
}
}

```

when

```

Result when(CDU eventname,
            CCB cb = NO_CALLBACK)

```

The Platform object receives all events at which time all callbacks registered for by the Platform objects are executed. Next, all of the callbacks registered for by image objects are executed, then all of the callbacks registered for by album objects are executed.

The preceding function call establishes a callback routine to handle an album-specific asynchronous event.

For example, you might want to know if any object was destroyed. You could say:

```

when("OBJECT_DESTROYED", Callback(destroyed_cb, 0)) ;

```

Caution – For the same album (albums having the same nickname are the same), multiple callbacks for the same event type are not supported.

The Album events shown in TABLE 3-9 are supported.

TABLE 3-9 Events Supported by Album

Events	Description
IMAGE_INCLUDED	An image was added to the album by some means. The new image is not automatically booted unless the AUTOIMAGE property was set. The <code>CurrentEvent::do_something()</code> function does nothing. This is an internal event, and has no MIS event info associated with it. Note: Use the method <code>Image::exists()</code> to see whether a given image from within IMAGE_INCLUDED exists.
IMAGE_EXCLUDED	An image was deleted from the album by some means or other. The <code>CurrentEvent::do_something()</code> function does nothing. Note that this is an internal event, and has no MIS event info associated with it.
OBJECT_CREATED	An object in the album came into existence. Call <code>CurrentEvent::do_something()</code> to cause inclusion in a tracking album before the end of the callback.
OBJECT_DESTROYED	An object in the album was destroyed. Call <code>CurrentEvent::do_something()</code> to cause exclusion in a tracking album before the end of the callback.
RAW_EVENT	Some album-related event occurred. You can examine it before the PMI does anything with it. The <code>CurrentEvent::do_something()</code> function does nothing.

Refer to the description of the `CurrentEvent::do_something` and `CurrentEvent::do_nothing` member functions in Section 3.15.3 “CurrentEvent Member Functions” on page 3-65” for more information on these `CurrentEvent` member functions.

3.10 AlbumImage Class

Inheritance: class AlbumImage

```
#include pmi/hi.hh
```

Data Members: No public data members are declared in this class.

An instance of the `AlbumImage` class is an iterator that represents the current album in a list of albums or the current image in a list of images. It can also be viewed as a two-way association between a given image and a given album.

3.10.1 Constructors

Default Constructor

```
AlbumImage( )
```

The default constructor creates an `AlbumImage` instance that refers to no actual `AlbumImage`. The value tests `FALSE` until you assign it a real `AlbumImage` value.

Copy Constructor

```
AlbumImage( AlbumImage& other)
```

This is an ordinary copy constructor. After the copy, both copies still refer to the same `AlbumImage` object. The reference count on the `AlbumImage` object is incremented.

3.10.2 Destructors

Default Destructor

```
~AlbumImage()
```

3.10.3 AlbumImage Operator Overloading

Assignment Operator

```
AlbumImage& operator = (const AlbumImage& other)
```

The assignment operator works like the copy constructor.

Cast Operator

```
operator void *()
```

The cast operator is for use in conditionals. It returns TRUE if this AlbumImage refers to an actual AlbumImage object. Do not attempt to use the returned value as a pointer to anything, since it points to private data.

Not Operator

```
int operator !()
```

This operator definition is provided so that you can say “if (!albumimage)...”

AlbumImage Class

Album *Operator*

```
operator Album()
```

This returns the album pointed to by the current AlbumImage association.

Image *Operator*

```
operator Image()
```

This returns the image pointed to by the current AlbumImage association.

3.10.4 AlbumImage Member Functions

This section describes the member functions of the AlbumImage class.

next_album

Purpose: Return the next album.

This is used when iterating over all of the albums of an image.

```
AlbumImage next_album()
```

next_image

Purpose: Return the next image.

This is used when iterating over all of the images of an album.

```
AlbumImage next_image()
```

3.11 AppTarget Class

Inheritance: public Album

Data Members: No public data members are declared in this class.

3.11.1 Constructors

```
AppTarget::operator !()
```

AppTarget is an abstraction that represents target applications.

3.11.2 AppTarget Operator Overloading

The ! is overloaded to check whether the Album is NULL or non NULL.

3.12 AuthApps Class

Inheritance: class AuthPriv

```
#include pmi/auth_apps.hh
```

Data Members: No public data members are declared in this class.

This class is derived from AuthPriv, which is an abstract base class for this class as well as the AuthFeatures class.

Note – AuthPriv is not documented since it is not directly used.

This class is used when you need to know which applications a user is authorized to use. Currently, this is used by the Launcher application to gray out the application icons which the user is not authorized to use.

After the application has successfully connected to the platform (using `Platform::connect`), an instance of this class is passed as an argument to the `Platform::get_authorized_apps` method. The method fills in the instance with the information about the applications the user is authorized to use. After the successful completion of this method, one can use the `AuthApps::is_authorized` method to find out whether an application is authorized or not. For more information, please refer to the description of the `Platform::get_authorized_apps` method.

3.12.1 Constructors

```
AuthApps ( )
```

The default constructor, above, creates an empty instance of this class which can be passed as an argument to the `Platform::get_authorized_apps` method.

3.12.2 AuthApps Operator Overloading

No public operators are defined for this class.

3.12.3 AuthApps Member Functions

The following are member functions for the `AuthApps` class.

`all_authorized`

```
Result all_authorized( ) const
```

This function determines if a given user has access to all applications or access to all the features of an application. This is an inherited function from the `AuthPriv` class.

`is_authorized`

```
Result is_authorized (const char *appname) const
```

Alternately,

```
virtual Result is_authorized (const char *appname,
const char *path) const;
```

This is an inherited function from the AuthPriv class. The function returns OK if the user is authorized to access the given application or its features. It should be called after successful completion of the Platform::get_authorized_apps function. Otherwise, it will always return NOT_OK.

3.13 AuthFeatures Class

Inheritance: class AuthPriv

```
#include pmi/auth_features.hh
```

Data Members: No public data members are declared in this class.

This class is derived from AuthPriv, which is an abstract base class for this class as well as the AuthApps class.

Note – AuthPriv is not documented because it is not directly used.

This class is used when you need to implement the feature level access control in your application. The application writer decides which features the application should have and what they control. The user who is running the application might not have access to all features. The application should query the list of features the user is authorized to use and accordingly restrict the operations the user can perform using the application.

After the application has successfully connected to the platform (using Platform::connect), an instance of this class is passed as an argument to the Platform::get_authorized_features method. This method fills in the instance with the information about the features the user is authorized to use. After the successful completion of the method, you can use the AuthFeatures::is_authorized method to find out whether a feature is authorized or not. For more information, please refer to the description of the Platform::get_authorized_features method. For an example program, please refer to \$EM_HOME/src/access_sample/access_feature_level.cc.

3.13.1 Constructor

```
AuthFeatures ()
```

The default constructor, above, creates an empty instance of this class which can be passed as an argument to the `Platform::get_authorized_features` method.

3.13.2 AuthFeatures Operator Overloading

No public operators are defined for this class.

3.13.3 AuthFeatures Member Functions

The following are member functions for the `AuthFeatures` class.

`all_authorized`

```
virtual Result all_authorized () const
```

This function determines if a given user has access to all applications or access to all the features of an application. This is an inherited function from the `AuthPriv` class.

`is_authorized`

```
virtual Result is_authorized (const char *feature) const
```

Alternately,

```
virtual Result is_authorized (const char *appname,  
const char *path) const;
```

This is an inherited function from the `AuthPriv` class. The function returns `OK` if the user is authorized to access the given application or its features. It should be called after successful completion of the `Platform::get_authorized_apps` function. Otherwise, it will always return `NOT_OK`.

3.14 Coder Class

Inheritance: `public Error`

```
#include pmi/hi.hh
```

Data Members: No public data members are declared in this class.

An instance of the `Coder` class specifies custom encoding and decoding functions for getting and setting the string value of a `Morf` or attribute. In general you would not call the methods of this class yourself. The PMI calls them when doing translation for various `get_str` and `set_str` operations. You set up these translation functions by deriving from the inner `CoderData` class, supplying virtual functions to do the appropriate translation, along with a virtual destructor to correctly destroy your `CoderData`. You can also declare other items in your derived class that are available to your routines.

The base `CoderData` class supplies an operator `Coder` to construct a `Coder` from a `CoderData`. Hence, the correct C++ incantation for creating a `Coder` is shown below.

```
Coder(*new OiCoderData(arg1, arg2...));
```

You then register the `Coder` with either `Platform::set_attr_coder` or `Syntax::set_coder`.

3.14.1 Constructors

Default Constructor

```
Coder()
```

The default constructor creates a `Coder` instance that refers to no actual `Coder`. The value tests `FALSE` until you assign it a real `Coder` value.

Copy Constructor

```
Coder(const Coder& other)
```

This is an ordinary copy constructor. After the copy, both copies still refer to the same `Coder` object. The reference count on the `Coder` object is incremented.

3.14.2 Coder Operator Overloading

Assignment Operator

```
Coder& operator = (const Coder& other)
```

The assignment operator works like the copy constructor.

Cast Operator

```
operator void*()
```

This cast operator is for use in conditionals. It returns `TRUE` if this `Coder` refers to an actual `Coder` object. Do not attempt to use the returned value as a pointer to anything, since it points to private data.

Not Operator

```
int operator!()
```

This operator definition is provided for conditionals such as:

```
if ( !albumimage ) ...
```

3.14.3 Coder Member Functions

This section describes the member functions of the Coder class.

get_str

Translate a Morf's value into a textual DU value.

```
DU get_str( Morf& mf, FBits fb )
```

set_str

Translate a textual DU value into a Morf's value.

```
Morf& set_str( Morf& mf, CDU data, FBits fb )
```

3.15 CurrentEvent Class

Inheritance: public Error

```
#include pmi/hi.hh
```

Data Members: No public data members are declared in this class.

CurrentEvent Class

An instance of the `CurrentEvent` class represents all the information that is known about an asynchronous event, available in a form that doesn't require the arbitrarily dangerous casting of various pointer values. A `CurrentEvent` is passed into every callback function, and is also returned by the `Waiter::wait` function. Within a callback, control is available both before and after any action that the PMI itself would perform on your behalf.

TABLE 3-10 CurrentEvent Method Types

Method Name	Method Type
<code>do_nothing</code> <code>do_something</code> <code>handled</code>	Control PMI performance
<code>get_event</code> <code>get_event_raw</code> <code>get_info</code> <code>get_info_raw</code> <code>get_message</code> <code>get_name</code> <code>get_oid</code>	Extract event information
<code>get_album</code> <code>get_eventtype</code> <code>get_image</code> <code>get_objclass</code> <code>get_objname</code> <code>get_platform</code> <code>get_time</code>	Extract contextual information
<code>something_to_do</code>	Set control information
<code>set_event_raw</code> <code>set_info_raw</code> <code>set_message</code> <code>set_name</code> <code>set_oid</code>	Set event information (called primarily by the PMI)
<code>set_album</code> <code>set_eventtype</code> <code>set_image</code> <code>set_objclass</code> <code>set_objname</code> <code>set_time</code>	Set contextual information (called primarily by the PMI)

3.15.1 Constructors

Default Constructor

```
CurrentEvent()
```

The default constructor creates a `CurrentEvent` instance that refers to no actual `CurrentEvent`. The value tests `FALSE` until you assign it a real `CurrentEvent` value.

Copy Constructor

```
CurrentEvent(const CurrentEvent& other)
```

This is an ordinary copy constructor. After the copy, both copies still refer to the same `CurrentEvent` object. The reference count on the `CurrentEvent` object is incremented.

Calldata Constructor

```
CurrentEvent( Ptr calldata)
```

This constructs a `CurrentEvent` from the *calldata* pointer passed into a callback as its second argument.

3.15.2 CurrentEvent Operator Overloading

Assignment Operator

```
CurrentEvent& operator = ( const CurrentEvent& other )
```

This assignment operator works like the copy constructor.

Cast Operator

```
operator void*()
```

This cast operator is for use in conditionals. It returns TRUE if this `CurrentEvent` refers to an actual `CurrentEvent` object. Do not attempt to use the returned value as a pointer to anything, since it points to private data.

Not Operator

```
int operator !()
```

This operator definition is provided for conditionals such as:

```
if ( !cur_event ) ...
```

3.15.3 CurrentEvent Member Functions

This section describes the member functions of the `CurrentEvent` class.

`do_nothing`

Purpose: Throw away the current event by removing callbacks from the `CurrentEvent` queue so that the PMI does nothing.

```
void do_nothing()
```

The PMI does not perform the action it would otherwise perform. This is meaningful only within a callback. If multiple objects receive callbacks for a given event (for example, one callback for an image, and one for the album containing the image), then each callback needs to call this function to disable the operations ordinarily done by the corresponding object.

Refer to TABLE 3-9 for information on how the member function `CurrentEvent::do_nothing` applies to the `Album::when` member function.

CurrentEvent Class

do_something

Purpose: Perform the customary action for this event by scheduling a callback in the CurrentEvent queue "something."

```
void do_something()
```

This is meaningful only within a callback. Before a callback is executed in response to an event, the PMI figures out what it wants to do and queues that operation using the `something_to_do` function. At the end of the callback, if you have not handled the event explicitly by calling either `do_something` or `do_nothing`, the PMI calls `do_something` on your behalf.

Refer to TABLE 3-9 for information on how the member function `CurrentEvent::do_something` applies to the `Album::when` member function.

get_album

Purpose: Return the associated album, if any.

```
Album get_album()
```

get_event

Purpose: Return, in *textual* form, the entire event message, if one exists.

```
DU get_event()
```

get_event_raw

Purpose: Return, in *encoded* form, the entire event message, if one exists.

```
Morf get_event_raw()
```

CurrentEvent Class

get_eventtype

Purpose: Return the type of event as specified by the `when` function that established the current callback.

```
DU get_eventtype()
```

get_image

Purpose: Return the image associated with this event, if one exists.

```
Image get_image(Boolean create = FALSE)
```

get_info

Purpose: Return, in *textual* form, the central event information.

```
DU get_info()
```

get_info_raw

The preceding function call returns in encoded form the central event information.

```
Morf get_info_raw()
```

get_message

Purpose: Return a pointer to the event message.

```
Ptr get_message()
```

This customarily returns a pointer to the event message, if one exists, though it could be used for any arbitrary data, depending on the event. You would not generally call this function unless you were in some fashion cheating on the PMI.

CurrentEvent Class

The returned pointer is unlikely to be meaningful outside the scope of the `CurrentEvent`, but if you're already cheating, you probably know what to do about that.

`get_name`

Purpose: Return the name of the event.

Every event has a unique OID associated with it in its GDMO definition. The `get_name` method returns the English name which corresponds to this OID (CommunicationsAlarm, for example). Use `get_oid` to return the OID.

```
DU get_name()
```

The `get_name` method returns the actual event name, which is not necessarily the same as the one used in the `when` function call that caused the callback to be invoked. Calling `get_eventtype` returns the internal event name used in the `when` function call, such as `RAW_EVENT`, whereas `get_name` returns a real event name such as `CommunicationsAlarm`.

`get_objclass`

Purpose: Return the name of the class of the associated image, if it exists.

```
DU get_objclass()
```

`get_objname`

Purpose: Return the name of the associated image, if it exists.

```
DU get_objname()
```

`get_oid`

Purpose: Return the OID.

CurrentEvent Class

The `get_oid` method returns the OID corresponding to the event name, if one exists. Internal events like `IMAGE_INCLUDED` have no OID.

```
Oid get_oid()
```

`get_platform`

Purpose: Return the platform.

This returns the MIS platform associated with this event.

```
Platform get_platform()
```

`get_time`

Purpose: Return, in ISO format, the time of the event, if available.

```
DU get_time()
```

`handled`

Purpose: Determine if the `CurrentEvent` was handled.

```
Boolean handled()
```

The `handled` method returns `TRUE` if and only if either of `do_something` or `do_nothing` has been called on the `CurrentEvent` for the current callback.

`set_album`

Purpose: Set the associated album.

```
void set_album( Album& al )
```

CurrentEvent Class

set_event_raw

Purpose: Set the encoded event message.

```
void set_event_raw( Morf& mf )
```

set_eventtype

Purpose: Set the event type.

```
void set_eventtype( CDU eventtype )
```

set_image

Purpose: Set the associated image.

```
void set_image( const Image& im )
```

set_info_raw

Purpose: Set the central event information.

```
void set_info_raw( const Morf& mf )
```

set_message

Purpose: Set the message pointer.

```
void set_message( Ptr msg )
```

CurrentEvent Class

set_name

Purpose: Set the event name.

```
void set_name( CDU nm )
```

set_objclass

Purpose: Set the object class.

```
void set_objclass( CDU cl )
```

set_objname

Purpose: Set the object name.

```
void set_objname( CDU nm )
```

set_oid

Purpose: Set the event OID.

```
void set_oid( Oid& o )
```

set_time

Purpose: Set the time of the event.

```
void set_time( CDU tm )
```

Error Class

something_to_do

Purpose: Queue up something to be done when do_something is called.

```
void something_to_do ( CCB cb, Ptr cdata )
```

Ordinarily the something_to_do method is called by the PMI before a callback is called, but you could use it to queue up additional operations to occur after the PMI calls do_something. It's usually easier, however, to call do_something yourself within the callback and then do whatever else needs doing.

3.16 Error Class

Inheritance: class Error

```
#include pmi/error.hh
```

The class Error stores details of errors related to an object instance of a derived-from-Error class, such as Image.

Example: If im1 is declared and used as follows:

```
Image im1 ;  
im1.fl() ;
```

...then if fl() fails, the error string and type can be retrieved with:

```
im1.get_error_string() ;  
im1.get_error_type() ;
```

For any function fl() of a class such as Image, if fl() is a static function, then no object is associated with that function. In such a case, you can use the variable error for error handling, where error is declared as:

```
Error error ;
```

Error Class

Example: `f2()` is a static function, in the class `Image`, called as follows:

```
Image::f2() ;
```

If `f2()` fails you can query errors as follows:

```
error.get_error_type() ;  
error.get_error_string() ;
```

3.16.1 Constructor

The syntax for the constructor for the `Error` class is:

```
Error(ErrorType etype=PMI_SUCCESS)
```

3.16.2 Error Operator Overloading

Assignment Operator

```
const Error &operator = ( const Error &err )
```

This assignment operator works like the copy constructor.

3.16.3 Error Public Data Member

The `Error` class has the following public data member:

```
static void (*error_entry_callback)(Error *)
```

The pointed-to callback function, if set, is called after entering a function of a class that is derived from `Error`. If that object instance is already in error, it might be because a previous function call had failed.

3.16.4 Error Member Functions

This section describes the member functions of the `Error` class.

`error_to_string`

Purpose: Return the default error string for a type.

```
static const char *error_to_string( ErrorType etype);
```

`get_error_string`

```
char    *get_error_string(void) const
```

When performing operations that interact with several objects in the MIS, be aware that the operations are "best effort" operations (where access control allows, the operation will succeed) and `get_error_string` may return a message which is confusing. For example, if a user is not allowed access to at least one object in an album, the user will receive the message, "Can't do this operation: access denied" (assuming the access control behavior is `denyWithResponse`), even though the given operation will succeed for all objects to which the user has access.

Error Class

get_error_type

```
ErrorType get_error_type(void) const
```

set_error_string

```
void set_error_string(char *)
```

set_error_type

```
void set_error_type(ErrorType)
```

set_error

```
void set_error(ErrorType, char *)
```

reset_error

```
void reset_error(void)
```

Note – All high level PMI calls must make use of the `reset_error` function to ensure that successive calls of the same type can succeed. For example, consider an `Image::send_event()` call which fails. If `reset_error` was not used in the call, successive `send_event` calls will fail with the same error message as the first, even if they would otherwise have succeeded.

```
set_error_entry_callback
```

```
static void set_error_entry_callback(void (*eec)(Error *)=0)
```

3.16.5 Error Types and Strings

TABLE 3-11 shows the error types and strings returned by `get_error_type()` and `get_error_string()`.

TABLE 3-11 Error Types

Type	Comment
MIS_ACCESS_NO_CONNECT_PRIVILEGE	Missing application connect privilege
MIS_ACCESS_USER_DOES_NOT_EXIST	Missing user profile
MIS_ACCESS_USER_NOT_MEMBER_OF_ANY_GROUP	User is not a member of any access control group
MIS_APP_INST_CREATE_FAILED	Failed to create application instance
MIS_CONNECTION_PDU_PARSING_FAILED	Failure during ape connect
MIS_CREATE_CALLBACK_FAILED	Failure of ape instance to create callback
MIS_ERROR	Error in MIS (unexpected?)
MIS_INVALID_PASSWORD	Invalid password on MIS host
MIS_RESOURCE_LIMIT	Ran out of memory
MIS_USER_DOES_NOT_EXIST	User does not exist on MIS host
PMI_CONNECTION_REPLY_PARSING_FAILED	Data passed by an application is not in an expected format
PMI_DATA_OBJECT_OP_FAILURE	Operation on associated data object failed
PMI_EM_LOGIN_DEAMON_PROBLEM	Check em_login daemon on MIS host
PMI_ENCODE_FAILED	Encoding of attribute failed
PMI_ERROR	Error
PMI_ILLEGAL_OPERATION	This operation cannot be performed on attribute

TABLE 3-11 Error Types (*Continued*)

PMI_INVALID_ARGUMENT	Argument is not valid in this context
PMI_MESSAGE_SENDS_FAILED	Sending of message failed
PMI_NEW_FAILED	New memory allocation failed
PMI_NO_DATA_OBJECT	Data object associated with this object is NULL
PMI_NOT_IMPLEMENTED	This feature is not implemented
PMI_NULL_ARGUMENT	Argument passed OR attribute used is NULL
PMI_OPERATION_FAILED	Some operation failed
PMI_SUCCESS	Success
PMI_UNKNOWN_PLATFORM	Platform is unrecognized
PMI_USER_ABORTED_CONNECTION	Canceled password dialog

3.17 Image Class

Inheritance: public Error

```
#include pmi/hi.hh
```

Data Members: No public data members are declared in this class.

An image is the local representation of a potential or actual framework object. If it represents an actual object, the image is capable of either manual or automatic synchronization. In many respects you can think of the image as the object itself, even though the actual object is off in the MIS, or even further away. Images give access to the object's methods and attributes.

Image Class

The image provides a model in which data is primarily textual, but also allows raw data to be passed in the form of `Morfs`. Images also provide attribute-like access to object and attribute schema information.

TABLE 3-12 Image Method Types

Member Function	Method Type
<code>first_album</code> <code>is_in_album</code> <code>num_albums</code>	Determine album membership
<code>find_by_nickname</code> <code>find_by_objname</code> <code>find_by_oi</code>	Global lookup
<code>boot</code> <code>shutdown</code> <code>start_boot</code> <code>start_shutdown</code>	Image Activation
<code>get_prop</code> <code>set_prop</code>	Control Properties
<code>get_userdata</code> <code>set_userdata</code>	User-defined information
<code>set_nickname</code> <code>set_objclass</code>	Set object information
<code>exists</code> <code>get_objclass</code> <code>get_nickname</code> <code>get_objname</code> <code>get_state</code> <code>get_oi</code> <code>get_oc</code> <code>get_encoded_oi</code>	Get object information
<code>attr_changed</code> <code>attr_exists</code> <code>get_attr_names</code> <code>get_attr_prop</code> <code>get_attr_trackmode</code> <code>set_attr_prop</code>	Attribute information
<code>get_attr_numerrors</code> <code>get_attr_last_error</code>	Get attribute error information

TABLE 3-12 Image Method Types (*Continued*)

Member Function	Method Type
set set_dbl set_from_ref set_gint set_long set_raw set_str	Set imaginary attribute values
get get_dbl get_gint get_long get_raw get_str	Get attribute values since last attribute value change event notification from the managed object
get_set get_set_dbl get_set_gint get_set_long get_set_raw get_set_str	Get imaginary attribute values (before they've been stored)
revert start_store store	Realize or discard imaginary attribute values
create start_create	Object creation, known name
create_within start_create_within	Object creation, known container
destroy start_destroy	Object destruction
call call_raw start start_raw	Miscellaneous object method activation
get_param_syntax get_result_syntax	Method data formats
get_when_syntax when	Notification

3.17.1 Image Constructor

The `Image` class is designed as a wrapper for a set of related classes. Invoking any of the constructors for the `Image` class creates an instance of this wrapper class. Note that no methods should be applied to this new image object until the `boot` method is applied, that is, the `Image` is booted. After booting, the image object is initialized and methods can be applied to it.

Default Constructor

```
Image ( )
```

The default constructor creates an image instance that refers to no actual image object. The value tests `FALSE` until you assign it a real image value.

Copy Constructor

```
Image(const Image& im)
```

This constructor is an ordinary copy constructor. After the copy, both copies still refer to the same image object.

General Constructor

```
Image ( CDU objname,
        CDU objclass = duNO_VALUE,
        Platform& plat = Platform::default_platform() )
```

This constructor creates an image instance for a particular kind of MIS. Because an image is really a wrapper for a set of related classes, this function actually works a bit like a virtual constructor.

The *objclass* is required to create an object, if the object does not exist. If such an object is creatable, then `boot ()` succeeds. If the object is not creatable, then `boot ()` fails. Refer to the description of `boot ()`.

If the object pointed to by *objname* already exists, the *objclass* parameter is not required. The *objclass* is populated in the object whenever a `get/set/delete` response is received from the MIS.

3.17.2 Image Operator Overloading

Assignment Operator

```
Image& operator = (const Image& other)
```

The *assignment* operator, above, works like the copy constructor.

Cast Operator

```
operator void*()
```

The *cast* operator, above, is for use in conditionals. It returns TRUE if this image refers to an actual image object. Do not attempt to use the returned value as a pointer to anything, since it points to private data.

Not Operator

```
int operator !()
```

The *not* operator, above, is provided so that you can say “if (!image)...”

3.17.3 Image Member Functions

This section describes the member functions of the Image class.

`attr_changed`

Purpose: Determine whether an attribute has changed.

```
Boolean attr_changed( CDU name )
```

This method returns TRUE if the *real* value of the named attribute was modified.

Image Class

Calling Sequence:

```
...
Image thing = Image(dn, clsnm);
CDU atnm = "abcXYZ" ; // Some valid attribute name for this class
...
if ( !thing.attr_changed( atnm ) ) {
    cout << "The attribute was modified.\n" ;
}
```

This notification refers only to a change reported in the *last received* notification from the MIS that an attribute was changed. The notification might be either an expected response to a store request of your own, or a somewhat-less-expected attribute change notification caused by someone else's store request. In the latter case, if you have asked for the ATTR_CHANGED notification, the image's real attributes do not change until the `CurrentEvent::do_something` function is called, either by you within the callback, or by the PMI after your callback returns. This allows you to get at the attribute values both before and after the change takes effect.

attr_exists

Purpose: Determine whether an attribute exists.

```
Boolean attr_exists( CDU name)
```

This function can return FALSE on an existing attribute if the value of EXISTS at that point is MAYBE.

Calling Sequence:

```
...
Image thing = Image(dn, clsnm);
CDU atnm = "abcXYZ" ; // Some valid attribute name for this class
...
if ( !thing.attr_exists( atnm ) ) {
    cout << "The attribute does not exist.\n" ;
}
```

Image Class

boot

Purpose: Activate an image and determine whether the object exists.

```
Result boot( Array (DU) attrlist,  
            Timeout to = DEFAULT_TIMEOUT)
```

Alternatively,

```
Result boot(const Timeout to = DEFAULT_TIMEOUT);
```

If the object does exist, any attributes not marked IGNORE become available for the various `get` functions. (If it does not exist, you can set attributes, then make a `create` or `create_within` function call.) If TRACKMODE was set to TRACK, the image also begins tracking changes to its object. When the boot is complete, the STATE of the image is set to UP.

If you pass `boot` without an attribute list argument, all attributes will be returned. If, however, you pass `boot` with an empty attribute list argument (an argument which contains an empty set of attributes), it will send a `get` request to the MIS to determine if the object exists, but no attribute values will be returned in the `get` response. In this way, you can check for the existence of an object without incurring the overhead of retrieving attribute values.

Calling Sequence:

```
...  
Image im(fdn);  
if( !im.boot() ) { // Boot an image without an attribute list.  
    printf("Error in boot\n");  
    return 2;  
}
```

For a complete example, refer to the `sample/image_boot.cc` file.

For an existing object, if the class specified at the time of object construction is incorrect, this is an error, and `boot()` fails. Refer to the description of the Image constructors.

Image Class

call

Purpose: Call a method for an image, with parameter as textual representation.

```
DU call( CDU name,  
         CDU param = duNONE,  
         const TIMEOUT to = DEFAULT_TIMEOUT)
```

Call a miscellaneous method for the managed object represented by this image. Both parameter and result are passed as textual data language.

In the syntax above, *name* is the name of the action. *param* is the parameter associated with the action. If *param* is not provided (or if it is set to duNONE), there is no parameter associated with this action specified in the GDMO.

The syntax of the parameter can be found using the `get_param_syntax` function. The syntax of the result is found using the `get_result_syntax` function. Action requests sent as a result of `call` are always sent with the confirmed bit set. To send unconfirmed action requests, please refer to `Image::start()`. This function (`Image::start()`) also sends unconfirmed action requests message when callback is `NO_CALLBACK`.

call_raw

Purpose: Call a method for an image, with parameter encoded.

Call a miscellaneous method for the managed object represented by this image. The parameter must be in encoded form.

```
Morf call_raw( CDU name,  
               Morf param = Morf(),  
               const TIMEOUT to = DEFAULT_TIMEOUT)
```

In the example above, *name* is the name of the action. *param* is the parameter associated with the action. If *param* is not provided (or if it is set to duNONE), there is no parameter associated with this action.

The syntax of the parameter can be found using `get_param_syntax`. The syntax of the result can be found using `get_result_syntax`. Action requests sent as a result of `call` are always sent with the confirmed bit set. To send unconfirmed action requests, please refer to `Image::start()`.

Image Class

create

Purpose: Create a managed object represented by a given image.

```
Result create( Image& refobj = Image(),
              const Timeout to = DEFAULT_TIMEOUT)
```

Calling Sequence:

```
Image thing = Image(dn,oc);
    before the next call, thing must be a valid image for an object. See create.cc
if ( !thing.create() ) {
    cout << "create failed" << endl;
    return 1;
} else {
    cout << "create succeeded" << endl;
}
```

You cannot create an object that already exists; however, you can check if an object already exists before creating it .

For a complete example, refer to the `sample/create.cc` file.

The attributes of the new object are a combination of the attributes supplied in the imaginary object, plus any additional attributes supplied by the reference object (if one was supplied), plus any other attributes the MIS feels like creating.

The `create` function requires the following:

- The object class of the image must be set correctly.
- There is sufficient information in the object name, nickname, and/or attributes for the MIS to figure out the complete object name for the new object (to the extent that it cannot simply make up a name).

After the `create`, the `OBJNAME` property has been set to the actual complete object name, even if it was not specified completely before the `create`.

create_within

Purpose: Create an object in a container.

Image Class

Within a specified container object, create, the managed object represented by the specified image.

```
Result create_within( CDU container_objname,  
                    Image& refobj = Image(),  
                    const Timeout to = DEFAULT_TIMEOUT)
```

For `create_within()`, first construct the image using the `duNone` constant as the instance name (instead of the `fdn`) as follows:

```
im = Image(duNone, object_class);  
im.create_within(args...);
```

Calling Sequence:

```
...  
Image thing = Image(duNone,oc);  
if ( !thing.create_within() ) {  
    cout << "create_within failed" << endl;  
    return 1;  
}
```

Typically you would use this when you want the MIS to make up a name for your object, and you only want to specify the object's location. You cannot create an object that already exists, or you receive an Invalid exception. The attributes of the new object are a combination of the attributes supplied in the imaginary object, plus any additional attributes supplied by the reference object (if one was supplied), plus any other attributes the MIS feels like creating.

The `create_within` function requires the following:

- The object class of the image must be set correctly.
- Sufficient information in the container object name plus the attributes for the MIS exist to determine the complete object name for the new object (to the extent that it cannot create a name).

After the `create`, the `OBJNAME` property is set to the actual complete object name, even though it was not specified before the `create`.

destroy

Purpose: Destroy the managed object represented by this image.

Image Class

Depending on the semantics of the MIS at that point, this might also delete any children of the object in question, or it might refuse to delete anything if there are any children.

```
Result destroy(const Timeout to = DEFAULT_TIMEOUT)
```

Calling Sequence:

```
...
Image thing = Image(dn,oc); // Valid image of an object.
...
if ( !thing.destroy() ) {
    cout << "destroy failed" << endl;
    return 1;
} else
    cout << "destroy succeeded" << endl;
}
```

exists

Purpose: Determine whether an object exists.

Given a booted image, determine whether a managed object exists.

```
Boolean exists()
```

Calling Sequence:

```
...
Image thing = Image(dn, clsnm);
...
thing.boot(); // Before you call exists(), must boot the image.
if ( !thing.exists() ) {
    cout << "The object does not exist.\n" ;
}
```

This function can return FALSE on an existing object before the first boot, since at that point the value of EXISTS is MAYBE.

Image Class

The following function call is equivalent to the above function call:

```
if ( !thing.get_prop("EXISTS") == "YES" ) {
```

find_by_nickname

Purpose: Find an image by its nickname.

Returns the value `Image()` if not found, which in a conditional evaluates to `FALSE`. All image nicknames are kept in a global registry.

```
static Image find_by_nickname( CDU name,  
                             Platform& plat = Platform::def_platform)
```

Calling Sequence:

```
...  
CDU nicnam = "Server" ;  
Image thing = Image::find_by_nickname( nicnam ) ;  
if ( !thing ) {  
    cout << "object not found" << endl;  
    return 1 ;  
}
```

find_by_oi

Purpose: Returns an image for the object identified by the argument *oi* - the object instance.

```
static Image find_by_oi(CDU oi, Platform$pl)
```

This static function is equivalent to the following member functions of the `Platform` class:

```
Image find_image_by_oi(oi)
```

If the object exists, it will be returned as a result of the function invocation. If the object does not exist the value `Image()` will be returned.

Image Class

find_by_objname

Purpose: Find an image by its object name.

Returns the value `Image()` if not found, which in a conditional evaluates to `FALSE`. All image object names are kept in a global registry.

```
static Image find_by_objname( CDU name,  
                             Platform& plat = Platform::def_platform)
```

Calling Sequence:

```
...  
CDU objnam = "Server" ;  
Image thing = Image::find_by_objname( objnam ) ;  
if ( !thing ) {  
    cout << "object not found" << endl;  
    return 1 ;  
}
```

To guarantee that the image is created if not found, use the `Image(objname)` constructor.

first_album

Purpose: Return the first album containing the image.

The return value is in the form of an `AlbumImage` iterator.

```
AlbumImage first_album()
```

The `AlbumImage::next_album` function returns a next album until there are no more.

get

Purpose: Return the value of the attribute,

Image Class

The attribute is formatted in data language, according to the implicit syntax of the attribute.

```
DU get( CDU name, FBits fb = 0) const
```

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;                // thing must be complete & booted.  
CDU atnam = "abcXYZ" ;        // Some existing attribute name  
CDU atval = thing.get( atnam ) ;
```

The syntax can specify either a list or a scalar. If the attribute is not present, a value of `DU()` is returned.

`get_attr_names`

Purpose: Get attribute names.

Return an array containing the names the attributes for this image. The image must be valid and booted before calling `get_attr_names()`.

```
Array DU get_attr_names( Boolean all = FALSE )
```

If *all* is:

- FALSE, returns the names of attributes that currently have values in the image.
- TRUE, returns the names of all attributes that are defined for the object class.

You can examine the `EXISTS` attribute property to see if the attribute exists in this image.

Calling Sequence:

```
thing.boot(); // thing must be a booted image before this next  
call.  
Array(DU) attr_names = thing.get_attr_names();
```

For a complete example, see the `sample/album.cc` file.

Image Class

get_attr_prop

Purpose: Get an attribute property.

Return a property key value of an attribute of the current image.

```
DU get_attr_prop( CDU attrname, CDU key)
```

Where, the *attrname* is an attribute name, the *key* is a property key, and the Return value is a property *key* value.

For instance, the property key TRACKMODE can have any one of the property key values IGNORE, SNAP, or TRACK. If *key* does not specify an existing property, a null DataUnit is returned, which tests FALSE in a conditional.

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;           // thing must be complete & booted.  
CDU atnm = "abcXYZ" ;     // Some valid, existing attribute name  
CDU propkey = "TRACKMODE" ; // Or: CDU propkey = duTRACKMODE  
DU propval = get_attr_prop( atnm, propkey ) ;
```

Some common property keys and values are listed in TABLE 3-13.

TABLE 3-13 Properties Supported by Most Image Attributes

Property Key	Values	Description
TRACKMODE	IGNORE , SNAP , TRACK	This tells the PMI how to track an attribute. Track attributes are maintained by monitoring attribute change events from the MIS. IGNORE attributes are assumed to be uninteresting and never fetched. Attempts to fetch an ignored attribute result in an invalid exception. In track mode, a PMI image is kept current based on Attribute Value Change notifications (AVCs). However, when tracking is off, AVCs are still issued but the image itself does not change
MOD_PENDING	IGNORE, REPLACE, INCLUDE, EXCLUDE , DEFAULT	This tells the PMI how to modify this attribute at the next store or start_store. IGNORE is the initial value, indicating that this attribute is not to be modified. REPLACE is the default set operation, and requests the MIS to do simple assignment using the supplied value. INCLUDE and EXCLUDE request the MIS to perform set inclusion or exclusion on a multi-valued attribute. (On a single-valued attribute, produces an invalid exception.) DEFAULT requests the MIS to set the attribute to its default value.
IGNORE_ALLO WED REPLACE_ALL OWED INCLUDE_ALL OWED EXCLUDE_ALL OWED DEFAULT_ALL OWED	YES , NO	These read-only properties say whether you can perform the corresponding modification to this attribute. They are meaningful only if the MIS supplies the corresponding information. It's possible for an attribute to claim to be modifiable and yet the MIS refuses to modify it.
MODIFIABLE	YES , NO	This read-only property is TRUE if any of REPLACE, INCLUDE, EXCLUDE, or DEFAULT are allowed.
EXISTS	YES , NO , MAYBE	This read-only property says whether this attribute is known to exist in the managed object represented by the image. Attributes with a TRACKMODE of IGNORE can remain in MAYBE state even though other attributes have been fetched.

Image Class

get_attr_trackmode

Purpose: Get the attribute TRACKMODE.

Return the TRACKMODE property value for an attribute of the current image.

```
DU get_attr_trackmode( CDU attrname )
```

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;           // thing must be complete & booted  
CDU atnm = "abcXYZ" ;     // Some valid, existing attribute name  
DU trkmd = thing.get_attr_trackmode( atnm ) ;
```

The following function call is equivalent to the one above:

```
DU trkmd = thing.get_attr_prop(atnm, "TRACKMODE") ;
```

get_attr_numerrors

Purpose: Get the number of attributes with outstanding errors.

Returns the number of attributes that have outstanding error messages resulting from the latest attempt to store.

```
U32 get_attr_numerrors()
```

Calling Sequence:

```
Image im ;  
U32 n ;  
..  
n = im.get_attr_numerrors() // The im image must be booted.
```

get_attr_last_error

Purpose: Get last error message for this attribute.

Image Class

Returns the error message associated with the last exception thrown by this attribute, or returned by the MIS.

```
DU get_attr_last_error(CDU attrname)
```

Calling Sequence:

```
Image im ;  
CDU atnm = "abcXYZ" ; // Some valid attribute name  
DU lem ;  
lem = im.get_attr_last_error( atnm ) ; // im must be booted.
```

get_dbl

Purpose: Get the attribute as a double.

Returns the value of the attribute formatted as a double. The **Syntax** must be a scalar and be consistent with a double representation. If the attribute is not present, a value of 0.0 is returned.

```
double get_dbl( CDU attrname )
```

Calling Sequence:

```
Image im ;  
CDU atnm = "abcXYZ" ; // Some valid attribute name  
double attrvalu = im.get_dbl( atnm ) ; // im must be booted.
```

get_gint

Purpose: Get the attribute as a GenInt.

Returns the value of the attribute formatted as a GenInt (arbitrarily long integer). The **Syntax** must be a scalar and must be consistent with a GenInt representation. If the attribute is not present, a value of GenInt is returned.

```
GenInt get_gint( CDU attrname )
```

Image Class

Calling Sequence:

```
Image im ;  
CDU atnm = "abcXYZ" ; // Some valid attribute name  
GenInt attrvalu = im.get_gint( atnm ) ; // im must be booted.
```

get_long

Get the attribute as a long.

Returns the value of the attribute formatted as a long. The Syntax must be a scalar and be consistent with a long representation. If the attribute is not present, a value of 0 is returned.

```
long get_long( CDU attrname )
```

Calling Sequence:

```
Image im ;  
CDU atnm = "abcXYZ" ; // Some valid attribute name  
long attrvalu = im.get_long( atnm ) ; // im must be booted.
```

get_nickname

Purpose: Get nickname of image.

Returns the nickname of the current image.

```
DU get_nickname()
```

The following function call is equivalent to the one above:

```
DU get_prop( "NICKNAME" ) ;
```

Calling Sequence:

```
Image im ;  
CDU ninm = im.get_nickname() ; // im must be booted.
```

Image Class

get_objclass

Purpose: Get class of object.

Returns the name of the class of the managed object represented by this image.

```
DU get_objclass()
```

The following function call is equivalent to the one above:

```
DU get_prop("OBJCLASS") ;
```

Calling Sequence:

```
Image im ;  
DU obcl = im.get_objclass() ; // im must be booted.
```

get_objname

Purpose: Get object name.

Returns the full name of the managed object represented by this image in local distinguished name (LDN) format.

```
DU get_objname()
```

The following is equivalent to the above expression:

```
DU get_prop("OBJNAME") ;
```

Calling Sequence:

```
Image im ;  
DU obnm = im.get_objname() ; // im must be booted.
```

Image Class

get_oi

Purpose: Get the Managed Object Instance, in encoded ASN1 value format, for the instance of the managed object represented by the Image.

```
Asn1Value get_oi()
```

get_encoded_oi

Purpose: Get the Managed Object Instance, in human readable format, for the instance of the managed object represented by the Image.

```
DU get_encoded_oi()
```

get_oc

Purpose: Get the Managed Object Class of the Managed Object Instance represented by the instance of the Image.

```
Asn1Value get_oc()
```

get_param_syntax

Purpose: Get the syntax for a method parameter.

```
Syntax get_param_syntax( CDU name )
```

For more information, refer to `call` and `start` under the description of Image.

Calling Sequence:

For the (arbitrarily chosen) object MDR and method `getOidName`:

```
Image im ( "metaname=\"MDR\"" ) ;  
im.boot() ;                               // im must be complete and booted.  
Syntax sntx ;  
sntx = im.get_param_syntax( DU ( "getOidName" ) ) ;
```

Image Class

get_prop

Purpose: Get a property of the current image.

```
DU get_prop( CDU key)
```

Where, the *key* is a property key *and Return value* is a property key value

For instance, If you specify the property TRACKMODE, and there is such a property, then the function returns one of the values IGNORE, SNAP, or TRACK.

If you specify a *key* that does not match an existing property, then the function returns a null DataUnit, which tests FALSE in a conditional.

Calling Sequence:

```
Image thing ;  
...  
thing.boot() ;           // thing must be complete and booted.  
CDU propkey = "TRACKMODE" ; // You can use duTRACKMODE  
DU propval = thing.get_prop( propkey ) ;
```

Most images support at least the following properties:

TABLE 3-14 Properties Supported by Most Images

Properties	Description
OBJNAME	The full name of the managed object represented by this image. To find an existing object, you must set either this property or the NICKNAME.
OBJCLASS	The name of the class of the managed object represented by this image. You must set this property in order to create an object.
NICKNAME	The nickname of this image.

TABLE 3-14 Properties Supported by Most Images (*Continued*)

Properties	Description
TRACKMODE (SNAP , TRACK)	How the PMI keeps track of the attribute value. SNAP images are fetched when the image is booted, and then left alone. TRACK images are maintained by monitoring events from the MIS. In track mode, a PMI image is kept current based on Attribute Value Change (AVC) notifications. However, when tracking is off, AVCs are still issued but the image itself does not change.
STATE (DOWN , BOOT , UP , SHUTDOWN)	The current state of the image. Read only. Initially, an image is DOWN. When you call <code>boot</code> or <code>start_boot</code> , the image enters BOOT state until the boot is done. The image then remains in the UP state until a shutdown or a <code>start_shutdown</code> is done. It remains in SHUTDOWN state until the shutdown is complete, at which time it is DOWN again.
EXISTS (YES , NO , MAYBE)	Whether the object as a whole is real or imaginary. Before the first <code>boot</code> , the state is indeterminate. Real objects cannot be created (unless you enjoy getting the Conflict exception). Imaginary objects can only be created or used as a reference object for a <code>set_from_ref</code> . Attempting to get a real attribute value from an imaginary object results in the <code>noshed</code> exception.

`get_raw`

Purpose: Get the attribute value, encoded form.

Returns the attribute value in an MIS-specific encoded form.

```
Morf get_raw( CDU attrname)
```

Calling Sequence:

```
Image thing ;
...
thing.boot() ;           // thing must be complete and booted.
CDU attrnam = "abcXYZ" ; // Some valid attribute name
Morf attrval = thing.get_raw( attrnam ) ;
```

Image Class

get_result_syntax

Purpose: Get the syntax for a method result.

```
Syntax get_result_syntax( CDU name )
```

Calling Sequence:

```
Image thing ;  
...  
thing.boot() ;           // thing must be complete and booted.  
CDU methnam = "abcXyz" ;   // Some valid method name  
Syntax sntx = thing.get_result_syntax( methnam ) ;
```

For more information, refer to `call` and `start` under the description of `Image`.

get_set

Purpose: Gets *imaginary* value from last set rather than *real* value.

The `get_set` function is like `get`, but it returns the *imaginary* value from the last set rather than the *real* attribute value.

```
DU get_set( CDU attrname, FBits fb = 0 )
```

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;           // thing must be complete & booted.  
CDU atnam = "abcXYZ" ;     // Some existing attribute name  
DU atval = thing.get_set( atnam ) ;
```

get_set_dbl

Purpose: Get *imaginary* double value from last set.

Image Class

The `get_set_dbl` function is like `get_dbl`, but it returns the *imaginary* value from the last set rather than the *real* attribute value.

```
double get_set_dbl( CDU attrname )
```

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;           // thing must be complete & booted.  
CDU atnam = "abcXYZ" ;    // Some existing attribute name  
dbl atval = thing.get_set_dbl( atnam ) ;
```

get_set_gint

Purpose: Like `get_gint`, but returns the *imaginary* value.

The `get_set_gint` function is like `get_gint`, but it returns the *imaginary* value from the last set rather than the *real* attribute value.

```
GenInt get_set_gint( CDU name )
```

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;           // thing must be complete & booted.  
CDU atnam = "abcXYZ" ;    // Some existing attribute name  
GenInt atval = thing.get_set_gint( atnam ) ;
```

get_set_long

Purpose: Like `get_long`, but returns the *imaginary* value.

The `get_set_long` function is like `get_long`, but it returns the *imaginary* value from the last set rather than the *real* attribute value.

```
long get_set_long( CDU name )
```

Image Class

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;                // thing must be complete & booted.  
CDU atnam = "abcXYZ" ;        // Some existing attribute name  
long atval = thing.get_set_long( atnam ) ;
```

get_set_raw

Purpose: Like `get_raw`, but returns the *imaginary* value from the last set.

The `get_set_raw` function call is like `get_raw`, but returns the *imaginary* value from the last set rather than the *real* attribute value.

```
Morf get_set_raw( CDU name )
```

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;                // thing must be complete & booted.  
CDU atnam = "abcXYZ" ;        // Some existing attribute name  
Morf atval = thing.get_set_raw( atnam ) ;
```

get_set_str

Purpose: Like `get_str`, but returns the *imaginary* value from the last set.

The `get_set_str` function is like an ordinary `get_str`, but returns the *imaginary* value from the last set rather than the *real* attribute value.

```
DU get_set_str( CDU name, FBits fb = 0 )
```

Image Class

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;           // thing must be complete & booted.  
CDU atnam = "abcXYZ" ;    // Some existing attribute name  
DU atval = thing.get_set_str( atnam ) ;
```

get_str

Purpose: Get attribute as a string.

Returns the value of the attribute formatted as a string without quotes.

```
DU get_str( CDU attrname, FBits fb = 0 )
```

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;           // thing must be complete & booted.  
CDU atnam = "abcXYZ" ;    // Some existing attribute name  
DU atval = thing.get_str( atnam ) ;
```

The **Syntax** must be a scalar and be consistent with a string representation. It is legal to get a numeric value as a string, it is converted for you. In fact, it's legal to `get_str` anything. If `get_str` doesn't know anything special to do with the data, it calls `get` for you.

The default format bits (0) sometimes produce strings containing newline characters. You might want to suppress this by passing an *fb* argument of `OMIT_NEWLINES`.

By default, the choice specifier is not returned as part of the string for the `get_str()` function. `USE_EXPLICIT_CHOICE` should be used as the second argument of this function if you want the choice specifier in the returned value.

If you have registered a `Coder` for this attribute (or in the absence of that, for the **Syntax** of this attribute), then that `Coder` is used to decode the attribute in preference to the standard decoder (the value need not be a scalar value in this case).

Image Class

get_userdata

Purpose: Get data the application stored under the *key* specified.

Returns any data the application might have stored under the *key* specified. There are no predefined values.

```
DU get_userdata( CDU key)
```

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;           // thing must be complete & booted.  
CDU atnam = "abcXYZ" ;    // Some existing attribute name  
DU atval = thing.get_userdata( atnam ) ;
```

If there is no data under that *key* for this instance, the return value (a null `DataUnit`) evaluates to `FALSE`.

get_when_syntax

Purpose: Get the syntax of a given event type.

```
Syntax get_when_syntax( CDU eventname)
```

Calling Sequence:

```
Image thing ;  
..  
thing.boot() ;           // thing must be complete & booted.  
CDU evnam = "abcXYZ" ;    // Some existing event name  
Syntax sntx = thing.get_when_syntax( evnam ) ;
```

For a list of events, refer to the description of *when*, under the description of the *Image* class.

is_in_album

Purpose: Determine whether the image is in the album.

Image Class

Returns TRUE if the image is contained in the album.

```
Boolean is_in_album( Album& album)
```

Calling Sequence:

```
Image thing ;  
Album myalbum ;  
...          // Before the next call, derive myalbum.  
DU atval = thing.is_in_album( myalbum ) ;
```

U32 num_albums

Purpose: Get the number of albums that contain this image.

```
U32 num_albums()
```

Calling Sequence:

```
Image thing ;  
thing.boot() ;  
...  
U32 na = thing.num_albums() ;
```

revert

Purpose: Cancel any pending sets.

Cancels any pending sets that have not yet been stored.

```
Result revert()
```

Calling Sequence:

```
Image thing ;  
thing.boot() ;  
...  
thing.revert() ;
```

Image Class

send_event

Purpose: Send event notifications to the MIS.

- Send an event to the MIS with a custom timestamp.

```
Result send_event (DU event_name,  
DU event_info=DU, struct tm *tt=0 )
```

- Send an event to the MIS with the timestamp expressed in ASN.1.

```
Result send_event (DU event_name,  
Asn1Value &event_info,  
Asn1Value &time=Asn1Value())
```

- Send an event to the MIS with the default timestamp of *now*.

```
Result send_event (DU event_name,  
struct tm *tt)
```

The image used in the `send_event()` call is the representation of the managed object that is generating the event notification (such as an alarm). This notification is sent to the MIS, from where it is forwarded to applications that are interested in events of this type.

The image instance (of which `send_event()` is a member) is the generator of the event. In using `send_event()`, a PMI application is acting in an agent role, since only agents generate events.

Before calling `send_event()`, keep in mind that:

- The image must be of a valid object class and have a valid name;
- The syntax of the argument `event_info` is based on the event and is parsed accordingly.

When the value of `tt` is zero, the event is sent with the timestamp of “Now”. Timestamp fields follow standard UNIX conventions:

```
Timestamp sent =  
(tt->tm_year + 1900, tt->tm_mon + 1, tt->tmday, tt->tm_hour,  
tt->tm_min, tt->tm_sec);
```

Image Class

set

Purpose: Set attributes.

The `set` function encodes the textual data you pass and modifies the value of the attribute using the encoded value.

```
Result set( CDU name, CDU val, CDU op = duREPLACE,  
           FBIts fb = 0 )
```

Actually, it changes the *imaginary* value of the attribute. The *real* value is not changed until the next `store`. The data language is interpreted according to the syntax already implicit in the attribute. If the data cannot be so interpreted, an invalid exception is thrown. The syntax can specify either a list or a scalar. A series of `set` operations can be undone before the `store` by calling `revert`. Refer to the `MOD_PENDING` attribute property in TABLE 3-13 for a description of legal operations.

set_attr_prop

Purpose: Set a property of an attribute.

Sets a property of an attribute of the current image.

```
Result& set_attr_prop( CDU name, CDU key, CDU value)
```

Requirements:

- *name* specifies an existing attribute in the object class
- *key* specifies a supported property
- *value* specifies a legal value.

If `set_attr_prop` cannot do what you ask, it throws an invalid exception. Refer to `get_attr_prop`, under the description of `Image`, for some typical properties.

set_dbl

Purpose: Modify an attribute using a double.

Encodes the double you pass and modifies the value portion of the attribute using the encoded value.

Image Class

Actually, it changes the *imaginary* value of the attribute. The *real* value is not changed until the next `store`. The syntax of the attribute must be a scalar and permit a double representation, or an invalid exception is thrown.

```
Result set_dbl( CDU name,  
               double val,  
               CDU op = duREPLACE)
```

Refer to the `MOD_PENDING` attribute property in TABLE 3-13 for a description of legal operations.

`set_from_ref`

Purpose: Copy attribute values from an object to its current image.

Copies the attribute values from the reference object into the current image.

```
Result set_from_ref( Image& refobj)
```

The state of the reference object must be `UP`, but the reference object need not exist. If the reference object exists, then its *real* attributes are copied. Otherwise its *imaginary* attributes are copied. Attributes that receive existing values are automatically given a `MOD_PENDING` property of `REPLACE`. Attributes that receive nonexistent values are given a `MOD_PENDING` property of `IGNORE`.

`set_gint`

Purpose: Modify an attribute using a `GenInt`.

Encodes the arbitrarily long integer you pass and modifies the value portion of the attribute using the encoded value. Actually, it changes the *imaginary* value of the attribute. The *real* value is not changed until the next `store`.

```
Result set_gint( CDU name,  
               GenInt& val,  
               CDU op = duREPLACE)
```

The syntax of the attribute must be a scalar and permit a `GenInt` representation, or an invalid exception is thrown. Refer to the `MOD_PENDING` attribute property in TABLE 3-13 for a description of legal operations.

Image Class

set_long

Purpose: Modify an attribute using a long.

Encodes the long you pass and modifies the value portion of the attribute using the encoded value. Actually, it changes the *imaginary* value of the attribute. The *real* value is not changed until the next store.

```
Result set_long( CDU name,  
                long val,  
                CDU op = duREPLACE)
```

The syntax of the attribute must be a scalar and permit a long representation, or an invalid exception is thrown.

Refer to the MOD_PENDING attribute property in TABLE 3-13 for a description of legal operations.

set_nickname

Purpose: Set the nickname for an image.

```
Result set_nickname( CDU nickname)
```

The following is equivalent to the above:

```
Result set_prop("NICKNAME", nickname) ;
```

set_objclass

Purpose: Set the object class for an image.

```
Result set_objclass( CDU name)
```

The following is equivalent to the above:

```
Result set_prop("OBJCLASS", name) ;
```

Image Class

set_prop

Purpose: Set a property of the current image.

Requirements:

- *key* specifies a supported property
- *value* specifies a legal value

```
Result set_prop( CDU key, CDU value)
```

If `set_prop` cannot do what you ask, it throws an invalid exception. Refer to `get_prop` under the description of the `Album` class, for some typical properties.

set_raw

Purpose: Load data into the *imaginary* value of an attribute.

The `set_raw` function loads a MIS-specific, encoded data value into the *imaginary* value of the attribute. If you later do a `store`, that updates the *real* value of the attribute.

```
Result set_raw( CDU attrname,  
               Morf& val,  
               CDU op = duREPLACE)
```

Refer to the `MOD_PENDING` attribute property in TABLE 3-13 for a description of legal operations.

set_str

Purpose: Encode a string and modify the attribute.

The `set_str` function encodes the string you pass and modifies the value portion of the attribute using the encoded value.

Image Class

Actually, it changes the *imaginary* value of the attribute. The *real* value is not changed until the next *store*. The syntax of the attribute must be a scalar and permit a string representation, or an invalid exception is thrown. For choice type attributes, a choice specifier should be used.

```
Result set_str( CDU attrname,
               CDU val,
               CDU op = duREPLACE, FBits fb = 0)
```

Calling Sequence:

```
Image im ;
im.boot() ;
char dn[300]          = "logId=\"AlarmLog\"";
char class_name[300]  = "log";
char attribute_name[300]= "maxLogSize";
char set_val[300]     = "666666";
...
if(!im.set_str(attribute_name, set_val)) {
...
}
```

The difference between `set_str` and `set` is that the data language used by `set` requires quotes as part of the string, while `set_str` assumes them if necessary. (They're not always necessary; you can also pass numeric values as strings, and they are converted for you.) Refer to the `MOD_PENDING` attribute property in TABLE 3-13 for a description of legal operations.

If you have registered a `Coder` for this attribute (or in the absence of that, for the `Syntax` of this attribute), then that `Coder` is used to encode the attribute in preference to the standard encoder—the value need not be a scalar value in this case.

set_userdata

Purpose: Store data supplied by the application.

The `set_userdata` function stores arbitrary data supplied by the application, under the *key* specified.

```
Result set_userdata( CDU key, CDU value)
```

There are no predefined values. If there was already data under that *key* for this instance, it is replaced without comment. Essentially, this is an associative array belonging to the album that the application can use any way it pleases.

Image Class

shutdown

Purpose: Deactivate an image.

The shutdown function deactivates an image and invalidates all locally cached attribute values.

```
Result shutdown( const Timeout to = DEFAULT_TIMEOUT)
```

A tracking image stops tracking. When complete, the image's STATE is set to DOWN.

start

Purpose: Provide an asynchronous version of call.

The start function is the asynchronous version of call. It sends an unconfirmed action request message when callback is NO_CALLBACK (the default).

```
Waiter start( CDU name, CDU param = duNONE, CCB cb = NO_CALLBACK)
```

start_boot

Purpose: Provide an asynchronous version of boot.

```
Waiter start_boot( CCB cb = NO_CALLBACK)
```

start_create

Purpose: Provide an asynchronous version of create.

```
Waiter start_create( Image& refobj = Image(),  
CCB cb = NO_CALLBACK);
```

```
virtual Result start_create( Image& refobj = Image(), CCB cb =  
NO_CALLBACK)
```

Image Class

start_create_within

Purpose: Provide an asynchronous version of `create_within`.

```
Waiter start_create_within( CDU container_objname,  
    Image& refobj = Image(),  
    CCB cb = NO_CALLBACK)
```

start_destroy

Purpose: Provide an asynchronous version of `destroy`.

```
Result start_destroy( CCB cb = NO_CALLBACK)
```

start_raw

Purpose: Provide an asynchronous version of `call_raw`.

```
Waiter start_raw( CDU name,  
    Morf param = Morf(),  
    CCB cb = NO_CALLBACK)
```

start_shutdown

Purpose: Provide an asynchronous version of `shutdown`.

```
Waiter start_shutdown( CCB cb = NO_CALLBACK);
```

start_store

Purpose: Provide an asynchronous version of `store`.

```
Waiter start_store( CCB &cb = NO_CALLBACK)
```

All `start_store` set requests are sent in unconfirmed mode when CCB is equal to `NO_CALLBACK` (the default).

Image Class

store

Purpose: Update actual attributes using imaginary attributes.

The `store` function updates the actual object's attributes using any imaginary attributes that have been created by `set` and any others.

```
Result store( const Timeout to = DEFAULT_TIMEOUT)
```

See also `start_store()`.

when

Purpose: Establish a callback routine.

The `when` function establishes a callback routine to handle an image-specific asynchronous event.

```
Result when( CDU eventname,  
            CCB cb = NO_CALLBACK)
```

The `Platform` object receives all events at which time all callbacks registered for by the `Platform` objects are executed. Next, all of the callbacks registered for by image objects are executed, then all of the callbacks registered for by album objects are executed.

For example, you might want to know if an attribute of the image changed. You might say:

```
when( "ATTR_CHANGED", Callback(attr_change_cb, 0)) ;
```



Caution – For the same image, multiple callbacks for the same event type are not supported.

Image Class

Image Events include:

TABLE 3-15 Image-specific Asynchronous Events

Events	Description
OBJECT_CREATED	The object represented by this image was successfully created (not necessarily by us!).
OBJECT_DESTROYED	The object represented by this image was successfully destroyed (not necessarily by us!).
ATTR_CHANGED	An attribute of the object represented by this image has changed in value.
RAW_EVENT	Any object-related event can be examined as a raw event before ordinary event processing by the PMI.

3.17.4 Related Global Functions

`fdn2formal`

Purpose: Take a fully distinguished name in "/" notation and convert it into "{...}" notation.

When passed a fully distinguished name with *fdn*, the `fdn2formal` function returns the "{...}" notation value of the FDN.

```
extern DU fdn2formal(CDU fdn);
```

`fdn2oi`

Purpose: Return the ASN1 encoded value for a fully distinguished name.

The `fdn2oi` function returns the encoded FDN if successful, otherwise NULL.

```
extern Asn1Value fdn2oi(CDU fdn);
```

Morf Class

name2oc

Purpose: Return the ASN1 encoded form of an object class, given the name in textual form.

The name2oc function returns the encoded object class if successful, otherwise NULL.

```
extern Asn1Value name2oc(DU oc);
```

oc2name

Purpose: Return the name of an object class.

If successful, the oc2name function returns the object class name.

```
extern DU oc2name(const Asn1Value& av);
```

oi2fdn

Purpose: Return the decoded FDN string, given the encoded FDN.

If successful, the oi2fdn function returns the decoded FDN string in "/" format.

```
extern DU oi2fdn(Asn1Value av);
```

3.18 Morf Class

Inheritance: public Error

```
#include pmi/hi.hh
```

Data Members: No public data members are declared in this class.

A Morf is a reference-counting wrapper around an abstract base class—MorfData. Each framework derives a new class from the base class and provides the implementation for manipulating that type of data in the context of the framework. Each instance of the derived class contains an opaque, encoded value along with the information necessary for the PMI to decode it.

Morf Class

In the context of Asn1 encoded values, an instance of a `Morf` class contains an `Asn.1` encoded value that can be retrieved as an `Asn1Value` instance through the `Morf` methods (e.g. `get_value`). The `Morf` class also contains an instance of the `Syntax` class, which specifies the type associated with the value.

As noted in their individual descriptions, some of this class’s methods require a list argument, some a scalar, and some accept both.

TABLE 3-16 Morf Method Types

Method Name	Data Domain	Method Type
<code>void*</code> <code>=</code> <code>==</code> <code>!=</code>		Operator Overloading
<code>get_platform</code> <code>get_syntax</code> <code>get_type</code> <code>has_value</code> <code>ref</code>	Defined type or list	Get information about an existing morf
<code>is_any</code> <code>is_choice</code> <code>is_list</code> <code>is_set</code> <code>is_sequence</code>	Defined type or list	Distinguish syntax types
<code>extract</code> <code>get_member_names</code> <code>num_elements</code> <code>split_array</code> <code>split_queue</code>	List only	Pull apart list morfs
<code>set</code>	Defined type or list	Set the data value of an existing morf

TABLE 3-16 Morf Method Types (Continued)

Method Name	Data Domain	Method Type
set_any set_dbl set_gint set_long set_str set_value	Defined type only	Set the data value of an existing morf
get() get_bit_string_identifiers() get_dbl get_gint get_long get_str get_size_constraint() get_value	Defined type only	Get the data value from an existing morf
	Defined type or list	
get_memname set_memname	Choice only	

3.18.1 Constructors

Morf()

The default constructor creates a Morf instance that refers to no actual morf. The value tests FALSE until you assign it a real morf value.

Morf(Syntax& syn)

The preceding constructor constructs a Morf instance for a particular kind of syntax. Since a Morf is reall a wrapper for a set of related classes, this function actually works like a virtual constructor. The newly constructed Morf will have an associated Syntax (passed as the argument), but no associated value.

Morf(const Morf& other)

Morf Class

The preceding constructor is a copy constructor. After the copy, both copies still refer to the same underlying `MorfData` object. The reference count on the morf object is incremented.

```
Morf(Syntax& syn, DU data)
```

The preceding constructor constructs a `Morf` instance for a particular kind of syntax, which implies a particular kind of MIS. Because a morf is really a wrapper for a set of related classes, this function actually works something like a virtual constructor. The textual data supplied as the second argument is parsed according to the syntax supplied, so you can create either defined type or list morfs with this function.

```
Morf(Syntax& syn, ArrayMorf& ma)
```

The preceding constructor constructs a `Morf` instance for a particular kind of syntax. Because a `Morf` is really a wrapper for a set of related classes, this function actually works rather like a virtual constructor. The newly constructed `Morf` will be associated with the specific `Syntax` that was passed as the first argument. Because an array of `Morf` is supplied as the second argument, only list `Morf` can be created with this function. The value associated with the newly constructed `Morf` will be constructed from the values of the `Morf` array elements, passed as the second argument.

```
Morf(Syntax& syn, class QueueMorfElem& mq)
```

The preceding constructor constructs a `Morf` instance for a particular kind of syntax. Because a `Morf` is really a wrapper for a set of related classes, this function actually works like a virtual constructor. The newly constructed `Morf` will be associated with the specific `Syntax` that was passed as the first argument. Because a queue of `MorfElems` is supplied as the second argument, only list `Morf` can be created with this function. The value associated with the newly constructed `Morf` will be constructed from the values of the `MorfElems`, passed as the second argument.

```
Morf( CDU attrname, Platform& plat = Platform::def_platform)
```

The preceding constructor constructs a `Morf` instance for a particular kind of attribute, which implies a particular syntax. The `attrname` passed as the first argument does not imply any specific instance of that attribute. Another way to accomplish this is to create an image for an object of the type containing the attribute

in question, and then extracting the `Morf` corresponding to that attribute. This constructor, however, can be used even when the attribute passed as the first argument is not associated with any specific class.

```
Morf( Ptr ptrdata, Boolean reuse = FALSE)
```

The preceding constructor constructs a `Morf` instance from a `void*` pointer created by the `ref` method. It is primarily for internal PMI use within callbacks, when the callback can occur after the original morf has gone out of scope, and would ordinarily have been deleted. Each call to `ref` increments a reference count, and each construction of a morf using this constructor eventually causes the reference count to be decremented again when the morf is destructed at the end of the callback. If multiple callbacks are to use the same pointer, then pass a `reuse` parameter of `TRUE` on all but the last callback (or call `ref` again within the callback) to keep the reference alive till the next callback.

3.18.2 Destructor

```
~Morf()
```

The destructor will decrement the reference count on the `MorfData` object, associated with the `Morf` object. If the reference count reaches 0, the destructor on the associated `MorfData` object is invoked.

3.18.3 Morf Operator Overloading

```
Morf& operator = (const Morf& other)
```

The assignment operator works like the copy constructor.

```
operator void*()
```

The preceding cast operator is for use in conditionals. It returns `TRUE` if this `Morf` refers to an actual morf object. Do not attempt to use the returned value as a pointer to anything, since it points to private data.

```
int operator !()
```

The preceding function is provided so that you can say “if (`!morf`)...”

```
int operator == (const Morf& other)
```

The preceding comparison operator returns `TRUE` if the two compared morfs are equivalent in value. The definition of “equivalent” depends on the MIS system type. Some values might be impossible to compare. The operator will test for equivalency of the values associated with the two compared morfs, if both values are interpreted to be of the specific type associated with the `Morf`.

Note – The type is implicitly associated with the `Morf` through the contained Syntax object.

```
int operator != ( const Morf& other)
```

The preceding comparison operator returns `TRUE` if the two compared morfs are *not* equivalent in value.

3.18.4 Morf Member Functions

This section describes the member functions of the `Morf` class.

`extract`

```
Morf extract( DU navigation)
```

The `extract` function extracts a morf from a tree of morfs. The *navigation* parameter is a string containing a dot-separated list of field names or position numbers. A position number (with the exception of 0) is an integer that represents the offset of a

morf, or element, in a list of morfs. The position number 0 returns the number of elements in the list at that level of the tree. In the case of Asn.1 encoded morfs, the following conventions are followed:

- If the `Asn1Type` associated with the `Morf` object is of type `SET` or `SEQUENCE`, then the navigation string contains a dot-separated list of field names, identifying the field as defined in the `SET` or `SEQUENCE`.
- If the `Asn1Type` associated with the `Morf` object is of type `SET OF` or `SEQUENCE OF`, then the navigation string contains a dot-separated list of position numbers, identifying the position in the `SET OF` or `SEQUENCE OF` structure.

For example, consider a morf, itself containing four morfs. If you call `extract` on the morf with the *navigation* parameter set to 0, the function will return a morf which can be checked over for the number of elements in the list.

Consider next a morf, itself containing five morfs, the third of which contains some arbitrary number of morfs, one of which happens to be called "attributevalue". To extract the morf "attributevalue", simply call `extract` on the original morf with the *navigation* parameter set to the string "3.attributevalue".

By manipulating the *navigation* parameter in this way, it is possible to refer to any morf in a tree of morfs. The string "3.6.2.attributevalue", for example, would refer to a morf called "attributevalue" that was four levels down in a tree of morfs.

The `extract` function is valid only for a morf that describes a list of values (or a choice). If called on a defined type morf it throws an invalid exception. When called on a choice value with a null *navigation* string, it returns a morf of the current inner type rather than the outer choice type.

Example: Use `extract` to find a component in a complex morf.

```
// If the object instance exists,
// get the value for attribute topoNodeMOSet,
// and construct a Morf.
Morf m = im.get_raw("topoNodeMOSet");
if (m.get_error_type() != PMI_SUCCESS) {
    cout << "Reason: ";
    cout << m.get_error_string() << endl;
    exit(5);
}

Morf newm = m.extract("1");
cout << "Using 1" << endl;
cout << newm.get_str().chp() << endl;
// Print out the attribute value
// before using Morf to split.

cout << "Using 1.distinguished " << endl;
newm = m.extract("1.distinguishedName");
cout << newm.get_str().chp() << endl;

cout << "Using 1.distinguishedName.1" << endl;
newm = m.extract("1.distinguishedName.1");
cout << newm.get_str().chp() << endl;

cout << "Using 1.distinguishedName.1.1" << endl;
newm = m.extract("1.distinguishedName.1.1");
cout << newm.get_str().chp() << endl;

cout << "Using 1.distinguishedName.1.1.attributeValue" << endl;
newm = m.extract("1.distinguishedName.1.1.attributeValue");
cout << newm.get_str().chp() << endl;
```

get

```
DU get(FBits fb = 0) const
```

The preceding function call returns the value of the morf formatted in data language according to the implicit Syntax of the morf. The Syntax can describe either a list or a defined type. Various format bits can be OR'ed together to influence the form of output. If the morf has no value, `DU()` is returned.

Morf Class

get()

```
Asn1Type get()
```

If applied on a `Morf` object that has the underlying syntax of an `ENUMERATED` type, this function returns the identifier string associated with the `ENUMERATED` value.

The same applies to the `Morf::get_str()` function.

CODE EXAMPLE 3-7 returns the identifier string associated with the enumerated value as defined in the `asn1` document.

CODE EXAMPLE 3-7 Morf::get() Example

```
Morf mrf;
// Assuming that morf is initialized to an ENUMERATED type
attribute
cout << "The value of the morf is " << mrf.get().chp() << endl;
```

get_bit_string_identifiers()

```
Result get_bit_string_identifiers(Array(Asn1NamedNumber) &idents)
const;
```

This method returns a reference to an array of type `Asn1NamedNumber`, which holds the identifiers and associated position for the `ASN.1 BIT STRING` type. If this method fails, it returns `NOT_OK`; otherwise, returns `OK`.

A sample program describing the use of the `get_bit_string_identifiers` method is in CODE EXAMPLE 3-8.

CODE EXAMPLE 3-8 Morf::get_bit_string_identifiers() Example

```
void show_idents(Array(Asn1NamedNumber) &idents) {
    Asn1TypeInt          int_type(AK_INTEGER);
    DU ident;
    GenInt  numbvalue;
    U32 i, orig_size;
    Asn1Value asn1number;
    Asn1ParsedValue number;

    orig_size = idents.size;
    cout << "Number of identifiers is " << orig_size << " ;" <<
    endl;
```


CODE EXAMPLE 3-8 Morf::get_bit_string_identifiers() Example (Continued)

```

        for (i = 0; i<orig_size; i++) {
            number = ids[i].num;
            ident = ids[i].name;
            if (number) {
                asnlnumber = number.get_real_val(int_type);
                asnlnumber.decode_int(numbvalue);
            }
            cout << "Identifier # =>" << i << " Name is => ";
            cout << ident.chp() << " ";
            cout << " Identifier position is => ";
            if (number) {
                cout << I32(numbvalue);
            } else {
                cout << "NULL";
            }
            cout << " ;" << endl;
        }
    }

void show_bit_string_identifiers(AsnlType rrrtype1) {
    Result rslt;
    Array(AsnlNamedNumber) newidents;

    rslt = rrrtype1.get_bit_string_identifiers(newidents);
    if (rslt == OK) {
        show_ids(newidents);
    } else {
        cout << "get_bit_string_identifiers returned NOT_OK!" <<
endl;
    }
}

```

get_dbl

```
double get_dbl()
```

The preceding function call returns the value of the morf formatted as a double. The Syntax must describe a defined type and be consistent with a double representation. Otherwise an invalid exception is thrown. If the morf has no value, 0.0 is returned.

Morf Class

get_gint

```
GenInt get_gint()
```

The preceding function call the value of the morf formatted as a `GenInt` (arbitrarily long integer). The `Syntax` must describe a defined type and be consistent with a `GenInt` representation. Otherwise an invalid exception is thrown. If the morf has no value, `GenInt()` is returned.

get_long

```
long get_long()
```

The preceding function call returns the value of the morf formatted as a `long`. The `Syntax` must describe a defined type and be consistent with a `long` representation. Otherwise an invalid exception is thrown. If the morf has no value, 0 is returned.

get_member_names

```
ArrayDU get_member_names()
```

The preceding function call returns the member names (field names) for a `list` value. For a `choice` value, it returns the member names of an inner list, presuming that the current value of the choice is a `list` value. To get the choice names themselves, you must use `get_syntax` and `get_member_names` instead.

get_memname

```
DU get_memname()
```

The preceding function call returns the name of the member (field) currently held by the morf. Valid only for members of lists and choices. (The morf itself need not be a list or choice.)

get_platform

```
Platform get_platform()
```

The preceding function call returns the Platform of the Syntax that is implicitly bound into the morf. Valid either for a morf that describes a defined type value or a list of values.

get_size_constraint()

```
Result get_size_constraint(Asn1ParsedValue &lower,
                           Boolean          &lower_open,
                           Asn1ParsedValue &upper,
                           Boolean          &upper_open
) const;
```

If this method is applied on an Asn1Type object that represents an ASN.1 type different from BIT STRING, OCTET STRING, SEQUENCE OF or SET OF, the method returns NOT_OK. If the invocation is successful, the method returns OK. In addition, this method returns, by reference, the lower and upper size constraints defined on a subtype of a BIT STRING, OCTET STRING, SEQUENCE OF, or SET OF base type as variables of type Asn1ParsedValue, and the lower_open and upper_open Boolean variables that specify whether the corresponding size constraint is open or closed. If the size constraint is open, the lower_open and upper_open variables are set to TRUE; otherwise, these two variables are set to FALSE.

The X.208 standard defines size constraints on BIT STRING, OCTET STRING, SEQUENCE OF, and SET OF types. In addition, the X.208 standard uses MIN and MAX to specify lower and upper constraints respectively.

MIN specifies the lower constraint defined for the parent type. MAX specifies the upper constraint defined for the parent type. The PMI library encodes MIN and MAX as NULL Asn1ParsedValue values. Therefore, after invoking the get_size_constraint method, you must check whether the returned values for the upper and lower size constraints are NULL, before attempting to decode them (see CODE EXAMPLE 3-9). CODE EXAMPLE 3-9 decodes and prints size constraints for BIT STRING, OCTET STRING, SEQUENCE OF, or SET OF types.

CODE EXAMPLE 3-9 Morf::get_size_constraint() Example

```
void show_size_constraint(Asn1Type type1) {
    char  buf1[5000], buf2[5000],
        *buf1p = buf1, *buf2p = buf2;
```

CODE EXAMPLE 3-9 Morf::get_size_constraint() Example (Continued)

```

U32    buf1len = 5000, buf2len = 5000;

Asn1TypeInt          int_type(AK_INTEGER);
Result rslt;
U32                  i;
Asn1Value  asn1lower, asn1upper;
Asn1ParsedValue  lower, upper;
GenInt low, up;
Boolean lower_open, upper_open;
rslt = typel.get_size_constraint(lower, lower_open, upper,
upper_open);
if (rslt == OK) {
    cout << "get_size_constraint was successfull!" << endl;
} else {
    cout << "get_size_constraint failed - NOT_OK!" << endl;
    return;
}

{
    if (lower) {
        asn1lower = lower.get_real_val(int_type);
        asn1lower.decode_int(low);

        buf2len = 5000;
        buf2p = buf2;
        int_type.format_value(asn1lower, buf2p, buf2len,
                               0, TAG_EXPLICIT, DataUnit(),
                               0);
        cout << "value of lower is "<<buf2<<" !"<<endl;
        // cout << "value of low is "<<I32(low)<<" !"<<endl;
        if (lower_open == TRUE) {
            cout << "lower range is open !" << endl;
        } else {
            cout << "lower range is closed !" << endl;
        }
    } else {
        cout << "value of lower is MIN !" << endl;
    }
}

{
    if (upper) {
        asn1upper = upper.get_real_val(int_type);
        asn1upper.decode_int(up);
        buf2len = 5000;
        buf2p = buf2;
        int_type.format_value(asn1upper, buf2p, buf2len,
                               0, TAG_EXPLICIT, DataUnit(),

```

CODE EXAMPLE 3-9 Morf::get_size_constraint() Example (*Continued*)

```

        0);
        cout << "value of upper is "<<buf2<<" !"<<endl;
        // cout << "value of up is "<<I32(up)<<" !"<<endl;
        if (upper_open == TRUE) {
            cout << "upper range is open !" << endl;
        } else {
            cout << "upper range is closed !" << endl;
        }
    } else {
        cout << "value of upper is MAX !" << endl;
    }
}
}

```

get_str

```
DU get_str( FBits fb = 0)
```

The preceding function call returns the value of the morf formatted as a string (without quotes). The Syntax must describe a defined type and be consistent with a string representation. It is legal to get a numeric value as a string; it is automatically converted for you. In fact, any value is legal. If the type is unrecognized, `get_str` calls `get` for you. If the morf has no value, `DU()` is returned.

The default format bits (0) sometimes produce strings containing newline characters. You might want to suppress this by passing an *fb* argument of `OMIT_NEWLINES`.

If you have registered a `Coder` for this morf's attribute (or in the absence of that, for the Syntax of this morf), then that `Coder` is used to decode the value in preference to the standard decoder. (The value need not be a defined type value in this case.) For more information, refer to `set_attr_coder` under the description of the `Platform` class, and `set_coder`, under the description of the `Syntax` class.

get_syntax

```
Syntax get_syntax()
```

The preceding function call returns the `Syntax` that is implicitly bound into the `Morf`. Valid either for a morf that describes a defined type value or a list of values.

Morf Class

get_type()

```
Asn1Type get_type()
```

This method returns the underlying `Asn1Type` object associated with a `Morf` object. If the `Morf` object is not initialized, the method returns a `NULL Asn1Type` (see CODE EXAMPLE 3-10).

CODE EXAMPLE 3-10 Morf::get_type() Example

```
Morf tstmrf = Morf();
Asn1Type tsttype = tstmrf.get_type();
if (tsttype) {
    cout << "The Asn1Type is initialized!" << endl;
} else {
    cout << "The Asn1Type is not initialized!" << endl;
}
```

get_value

```
Asn1Value get_value()
```

The preceding function call returns the encoded value stored in the `morf`, if any. No attempt is made, at this point, to validate the value against the type associated with the `morf`.

has_value

```
void* has_value()
```

The preceding function call returns a pointer to the internal, MIS-specific value, if any. Otherwise, returns 0. Note that this differs from operator `void*()`, which can return `TRUE` even when the `morf` contains only a `Syntax` with no value.

Morf Class

is_any

```
Boolean is_any()
```

This function returns `TRUE` if the type of the data is of the type "ANY DEFINED BY" from the GDMO definition of the object.

is_choice

```
Boolean is_choice()
```

The preceding function call returns `TRUE` if the Syntax of the morf describes a choice value. A choice value can have any one of a number of types of value. (The `get_memname` function tells you which kind of value the current morf is holding.) Valid either for a morf that describes a defined type value or a list of values.

is_list

```
Boolean is_list()
```

The preceding function call returns `TRUE` if the Syntax of the morf describes a compound data value, and `FALSE` if it describes a defined type value. Valid either for a morf that describes a defined type value or a list of values. Note that a list with only one element, or zero elements, is still a list and not a defined type.

is_sequence

```
Boolean is_sequence()
```

The preceding function call returns `TRUE` if the type of the data describes a compound data value that contains an ordered list of zero or more members.

Morf Class

is_set

```
Boolean is_set()
```

The preceding function call returns TRUE if the type of the data describes a compound data value that contains an unordered list of zero or more members.

num_elements

```
U32 num_elements()
```

The preceding function call returns the number of elements in the morf's list. Valid only for a morf that describes a list of values.

ref

```
Ptr ref()
```

The preceding function call returns a reference-counted (refcnt) void* pointer to this morf, from which you *must*, at some future time, reconstruct the morf using the `Morf(Ptr, Boolean)` constructor. Refer to that constructor description earlier in this section for further information.

set

```
Morf set( CDU data, Fbits fb = 0)
```

The preceding function encodes the textual data you pass and replaces the value portion of the morf with the encoded value. The data language is interpreted according to the `Syntax` already implicit in the morf. If the data cannot be so interpreted, an invalid exception is thrown. The `Syntax` can describe either a list or defined type.

Morf Class

set_any

```
Morf set_any( Morf& data)
```

This function returns TRUE if the type of the data contained in the morf is of the type "ANY DEFINED BY" from the GDMO definition of the object. The morf data is provided in *data*.

set_dbl

```
Morf set_dbl( double data)
```

The preceding function call encodes the double you pass and replaces the value portion of the morf with the encoded value. The *Syntax* of the morf must be a defined type and should permit a double representation, or an invalid exception is thrown.

set_gint

```
Morf set_gint(GenInt& data)
```

The preceding function call encodes the GenInt you pass and replaces the value portion of the morf with the encoded value. (A GenInt is an arbitrarily long integer.) The *Syntax* of the morf must be a defined type and permit a GenInt representation, or an invalid exception is thrown.

set_long

```
Morf set_long(long data)
```

The preceding function call encodes the long you pass and replaces the value portion of the Morf with the encoded value. The *Syntax* of the morf must be a defined type and should permit a long representation, or an invalid exception is thrown.

Morf Class

set_memname

```
Result set_memname(CDU name)
```

The preceding function call sets the name of the member (field) currently held by the morf. Valid only for members of a choice type. The old value of the morf is discarded.

set_str

```
Morf set_str( CDU data, FBits fb = 0 )
```

The preceding function call encodes the string you pass and replaces the value portion of the morf with the encoded value. The `Syntax` of the morf must be a defined type and permit a string representation, or an invalid exception is thrown.

The difference between `set_str` and `set` is that the data language used by `set` requires quotes as part of the string, while `set_str` assumes them if necessary. (They're not always necessary; you might also pass numeric values as strings, and they are converted for you.)

If you have registered a `Coder` for this morf's attribute (or in the absence of that, for the `Syntax` of this morf), then that `Coder` is used to encode the value in preference to the standard encoder. (The value need not be a defined type value in this case.) For more information, refer to `set_attr_coder` under the description of the `Platform` class, and `set_coder`, under the description of the `Syntax` class.

set_value

```
void set_value(Asn1Value& data)
```

The preceding function call sets the encoded value into the morf. Checking is not performed on the value (so that it is not validated, at this point, against the type associated with the morf).

`split_array`

```
Array Morf split_array()
```

The preceding function call returns the elements of a list morf in `Array` form, such that they can be indexed numerically.

Valid only for a morf that describes a list of values. If called on a defined type morf, it throws an invalid exception.

`split_queue`

```
class :/Queue MorfElem split_queue()
```

The preceding function call returns the elements of a list morf in `Queue` form, such that they can be processed with ordinary `Queue` commands. Valid only for a morf that describes a list of values. If called on a defined type morf it throws an invalid exception.

3.19 MorfBuilder Class

Inheritance: None

```
#include extpmi/exthi.hh
```

Data Members: No public data members are declared in this class.

The `MorfBuilder` class is a utility wrapper built on top of the `Morf` class. `MorfBuilder` provides more flexibility in dealing with `Morf` objects that represent constructed types (`SET`, `SET OF`, `SEQUENCE`, and `SEQUENCE OF`) and minimizes the amount of coding you have to do.

For example, using `MorfBuilder` objects, you can create empty `Morf` objects for a constructed type and incrementally update the fields of the underlying `Morf` object. This functionality is not supported at the `Morf` object level.

3.19.1 Constructors

```
MorfBuilder ( Morf& morf )
```

The preceding constructor creates a `MorfBuilder` instance that contains a `Morf` object and implicitly refers to that `Morf` object in most `MorfBuilder` methods. Initially, the `Morf` object contained in the new `MorfBuilder` instance has the same syntax and `Asn1Value` as *morf*. The syntax of the underlying `Morf` object does not change throughout the life of the `MorfBuilder` object. The underlying `Morf` object's `Asn1Value`, however, can be modified using the `MorfBuilder` object's methods.

```
MorfBuilder (Syntax& syn )
```

This constructor creates a `MorfBuilder` instance that contains a `Morf` object and implicitly refers to this `Morf` object in most `MorfBuilder` methods. Initially, the `Morf` contained in the new `MorfBuilder` instance has the same syntax as *syn* and an empty `Asn1Value`.

The syntax of the underlying `Morf` object does not change throughout the life of the `MorfBuilder` object. The underlying `Morf` object's `Asn1Value`, however, can be modified using the `MorfBuilder` object's methods.

```
MorfBuilder (const MorfBuilder& old_mbd )
```

The preceding constructor is a copy constructor. After the copy operation is completed, the newly created `MorfBuilder` implicitly refers to a `morf`, that has the same `Syntax` and `Asn1Value` as the `morf` associated with the `MorfBuilder` instance being copied. Both instances of the `MorfBuilder` class are independent copies, and both refer to two separate instances of the `Morf` class. Updates to the `Asn1Value` of one of the `MorfBuilder` instances do not affect the `Asn1Value` of the other `MorfBuilder` instance.

```
MorfBuilder (CDU attrname, Platform& plat = Platform::def_platform)
```

The preceding constructor creates a `MorfBuilder` instance for the given kind of attribute which implies a particular syntax.

3.19.2 Destructor

```
~MorfBuilder ()
```

This destructor releases all the resources associated with the `MorfBuilder`.

3.19.3 MorfBuilder Operator Overloading

```
MorfBuilder& operator = (const MorfBuilder& other)
```

The assignment operator works like the copy constructor.

```
operator void*()
```

The preceding cast operator is to be used in conditionals. It returns `TRUE` if the `MorfBuilder` refers to an actual `Morf` object.



Caution – Do *not* attempt to use the returned pointer value because it points to private data.

```
int operator !()
```

The preceding function is a logical negation that can be used in an “if (`!morfbuilder`)...” context.

3.19.4 MorfBuilder Member Functions

This section describes the member functions of the `MorfBuilder` class.

get_raw

```
Morf get_raw (int do_assemble = TRUE);
```

If *do_assemble* is TRUE, this function updates the cached overall internal Asn1Value/Syntax relation by assembling the Asn1Value/Syntax member values of the underlying morf into a new morf.

Updating the cached overall internal Asn1Value/Syntax relation is needed when MorfBuilder is associated with a constructed ASN.1 type such as SEQUENCE, SEQUENCE OF, SET, and SET OF. In such a case, you can update the member values of the underlying morf, but the overall internal Asn1Value /Syntax relation is not updated until you use the get_raw method.

On successful completion, this function returns the newly assembled Morf. Otherwise, the underlying Morf is not updated and an empty Morf is returned.

Therefore, you should check the returned Morf to determine whether it is empty. For example, !morfbuilder.get_raw().

If *do_assemble* is FALSE, this function returns the underlying Morf without assembling a new Morf. Invoking get_raw(FALSE) does not reset any modifications that might have been done to the MorfBuilder. These modifications are still kept and can be assembled into a modified Morf by calling get_raw(TRUE) at a later time. MorfBuilder always keeps a cached copy of the latest successfully assembled valid Morf, and updates the cached copy with a new Morf only if the new Morf is successfully assembled by invoking get_raw(TRUE).

```
Morf get_raw (CDU navigation, int do_assemble = TRUE);
```

Used with a MorfBuilder that is associated with a constructed ASN.1 type, this function recursively navigates to the specified morf in a MorfBuilder using the given navigation string and calls get_raw (*do_assemble*).

The *navigation* parameter is a dot-separated list of field names or position numbers. A position number, with the exception of 0, is an integer that represents the offset of a Morf object in a list of Morf objects.

This function returns the Morf object that is associated with the MorfBuilder at the field described by the navigation string. It splits the underlying MorfBuilder into its component parts (for example, a tree of MorfBuilder classes) and recursively invokes the get_raw (newnavigation, *do_assemble*) function, where the newnavigation argument is the old navigation argument, stripped from the first component of the dot separated list, until one entry is left in the navigation string. This means that the function reached the Morf specified in the navigation string. This function then calls the get_raw (*do_assemble*) function.

MorfBuilder Class

If the MorfBuilder does not represent a constructed type, or the navigation string does not represent a valid field of the asn1 type, this function returns an empty Morf object.

CODE EXAMPLE 3-11 shows how to use `get_raw` to find a component in a complex morfbuilder.

CODE EXAMPLE 3-11 Using `get_raw` With a Complex morfbuilder

```
/*
SEQUENCE {
    seqint INTEGER DEFAULT 10,
    seqchar1 OCTET STRING OPTIONAL,
    seqbool1 BOOLEAN,
    seqseqof SEQUENCE OF OCTET STRING
}

If the syntax
Syntax syn
is associated with the above described SEQUENCE type,
and the morfbuilder
MorfBuilder mbd
is associated with the syntax syn
*/

    Morf m1 = mbd.get_raw("seqint");
    if (!m) {
        cout << "Failed to get the morf field!" << endl;
    }
// returns the Morf associated with the field seqint

    Morf m2 = mbd.get_raw("seqseqof.2");
    if (!m2) {
        cout << "Failed to get the morf field!" << endl;
    }
// returns the Morf associated with the field seqseqof.2 - i.e.
// the second field of the SEQUENCE OF seqseqof

    Morf m3 = mbd.get_raw(FALSE);
    if (!m3) {
        cout << "Failed to get the old morf!" << endl;
    }
// returns the Morf associated with the MorfBuilder
// without attempting to reassemble the Morf, if fields of the
// morf were modified
```

CODE EXAMPLE 3-11 Using get_raw With a Complex morfbuilder (Continued)

```

        Morf m4 = mbd.get_raw(); // default argument is TRUE
        if (!m4 {
            cout << "Failed to assemble and get the new morf!" << endl;
        }
    // returns the Morf associated with the MorfBuilder
    // first attempts to reassemble the Morf, if fields of the
    // morf were modified

```

select_choice

```
Result select_choice (CDU member_name);
```

Selects a syntax for the MorfBuilder associated with a CHOICE type specified in the *member_name* argument.

Returns OK on successful completion; otherwise, returns NOT_OK.

Note – This function should be applied *only* to MorfBuilder instances that represent CHOICE ASN.1 types. If not, this function returns NOT_OK. Also, if the specific type of a MorfBuilder representing a CHOICE type has been already set, either through a previous invocation of select_choice or through initialization with a Morf object, this function returns NOT_OK.

```
Result select_choice (CDU navigation, CDU member_name);
```

Used with a MorfBuilder that is associated with a constructed ASN.1 type, this function recursively navigates to the specified Morf in a MorfBuilder using the given navigation string (*navigation*) and calls select_choice (*member_name*).

The *navigation* parameter is a dot-separated list of field names or position numbers. A position number, with the exception of 0, is an integer that represents the offset of a Morf object in a list of Morf objects.

If the MorfBuilder does not represent a constructed type, or the navigation string does not represent a valid field of the ASN.1 type, the function returns NOT_OK.

set_syntax

```
Result set_syntax (Syntax& new_syn);
```

Selects a syntax for a MorfBuilder associated with a CHOICE type in the *new_syn* argument.

Returns OK on successful completion; otherwise, returns NOT_OK.

Note – This function should be applied *only* to MorfBuilder instances that represent CHOICE ASN.1 types. If not, this function returns NOT_OK. Also, if the specific type of a MorfBuilder representing a CHOICE type has already been set, either through a previous invocation of select_choice or through initialization with a Morf object, this function returns NOT_OK.

```
Result set_syntax (CDU navigation, Syntax& new_syn);
```

Used with a MorfBuilder that is associated with a constructed ASN.1 type, this function recursively navigates to the specified Morf in a MorfBuilder using the given navigation string (*navigation*) and calls set_syntax (*new_syn*).

The *navigation* parameter is a dot-separated list of field names or position numbers. A position number, with the exception of 0, is an integer that represents the offset of a Morf object in a list of Morf objects.

If the MorfBuilder does not represent a constructed type, or the navigation string does not represent a valid field of the ASN.1 type, the function returns NOT_OK.

set_raw

```
Result set_raw (Morf& morf, FBits fb = 0);
```

Replaces the MorfBuilder object's underlying Morf object and any contained MorfBuilder objects with *morf*. This means that if the MorfBuilder represents a constructed type and parts of it have been previously updated, these updates are lost.

Note – To avoid losing previous updates to the member values of a MorfBuilder object that represents a constructed type, use the `get_raw` function before using the `set_raw` function.

Returns OK on successful completion; otherwise, returns NOT_OK.

```
Result set_raw (CDU navigation, Morf& morf, FBits fb = 0);
```

Used with a MorfBuilder that is associated with a constructed ASN.1 type, this function recursively navigates to the specified Morf in a MorfBuilder using the given navigation string (*navigation*) and calls `set_raw (morf, fb)`.

The *navigation* parameter is a dot-separated list of field names or position numbers. A position number, with the exception of 0, is an integer that represents the offset of a Morf object in a list of Morf objects.

If the MorfBuilder does not represent a constructed type, or the navigation string does not represent a valid field of the ASN.1 type, the function returns NOT_OK.

set

```
Result set (CDU value, FBits fb = 0);
```

This function replaces the MorfBuilder's underlying morf value and any contained MorfBuilder objects with *value*. This means that if the MorfBuilder represents a constructed type and parts of it have been previously updated, these updates are lost.

Note – To avoid losing previous updates to the member values of a MorfBuilder object that represents a constructed type, use the `get_raw` function before using the `set` function.

Returns OK on successful completion; otherwise, returns NOT_OK.

```
Result set (CDU navigation, CDU value, FBits fb = 0);
```

Used with a MorfBuilder that is associated with a constructed ASN.1 type, this function recursively navigates to the specified Morf in a MorfBuilder using the given navigation string (*navigation*) and calls `set (morf, fb)`.

MorfBuilder Class

The *navigation* parameter is a dot-separated list of field names or position numbers. A position number, with the exception of 0, is an integer that represents the offset of a Morf object in a list of Morf objects.

If the MorfBuilder does not represent a constructed type, or the navigation string does not represent a valid field of the ASN.1 type, the function returns NOT_OK.

validate

```
Result validate (Boolean ignore_tag = FALSE);
```

Validates the MorfBuilder's underlying Morf object. It does that by invoking internally the function `get_raw(TRUE)` on the underlying Morf object. If the `get_raw(TRUE)` operation fails, the `validate` function returns NOT_OK. Otherwise, the `validate` function continues. It validates the Asn1Value of the Morf, against the associated ASN.1 type by invoking the method `Asn1Type::validate(Asn1Value value, Boolean ignore_tag)`.

Returns OK on successful completion; otherwise, returns NOT_OK.

```
Morf validate (CDU navigation, Boolean ignore_tag = FALSE);
```

Used with a MorfBuilder that is associated with a constructed ASN.1 type, this function recursively navigates to the specified Morf in a MorfBuilder using the given navigation string (*navigation*) and calls `validate (ignore_tag)`.

The *navigation* parameter is a dot-separated list of field names or position numbers. A position number, with the exception of 0, is an integer that represents the offset of a Morf object in a list of Morf objects.

If the MorfBuilder does not represent a constructed type, or the navigation string does not represent a valid field of the ASN.1 type, the function returns NOT_OK.

get_prop

```
DU get_prop (CDU key);
```

The preceding function returns the property addressed by the key *key*. The properties are local to the MorfBuilder class and are used to control some of its behavior. TABLE 3-17 lists the currently defined properties for the MorfBuilder class.

TABLE 3-17 Properties Addressed by *key*

Key	Values
access_type	by_index
	by_name (default)

The access_type property is relevant only to a MorfBuilder class that represents a SEQUENCE type.

When the access_type property is set to by_name, all the MorfBuilder class functions that use the *navigation* string argument interpret the string as the member name of the SEQUENCE to be accessed.

When the access_type property is set to by_index, all the MorfBuilder functions that use the *navigation* string argument interpret the string as the position index for the member of the SEQUENCE to be accessed.

For example, suppose a MorfBuilder object represents an object whose type is the SEQUENCE (see CODE EXAMPLE 3-12).

CODE EXAMPLE 3-12 SEQUENCE Type Example

```
SEQUENCE {  
    int INTEGER,  
    char OCTET STRING  
}
```

If the access_type property of the MorfBuilder object is set to by_name, members of the SEQUENCE type can be accessed by their names (for example, int and char).

MorfBuilder Class

If, however, the `access_type` property of the `MorfBuilder` object is set to `by_index`, members of the `SEQUENCE` type can be accessed by their position indexes (for example, 1 and 2). CODE EXAMPLE 3-13 shows how to use the `get_prop()` function.

CODE EXAMPLE 3-13 `get_prop()` Example

```
Morf morf;
if (mbd.get_prop("access_type") == DU("by_name")) {
    cout << "access_type is by_name !" << endl;
    morf = mbd.get_raw("char");
} else if (mbd.get_prop("access_type") == DU("by_index")) {
    cout << "access_type is by_index !" << endl;
    morf = mbd.get_raw("2");
} else
    cout << "Invalid access_type is bad !" << endl;
```

```
DU get_prop (CDU navigation, CDU key);
```

Invokes `get_prop(CDU key)` on the sub-`MorfBuilder` member addressed by the `navigation` string.

`get_error_type`

```
ErrorType get_error_type(void) const;
```

This function enables a `MorfBuilder` object to maintain the error status that results from the invocation of a `MorfBuilder` function until another function is invoked on the same `MorfBuilder`. The returned error type can be retrieved through the `get_error_type()` function.

For example, suppose a `MorfBuilder` object represents an object whose type is `SEQUENCE` (see CODE EXAMPLE 3-14).

CODE EXAMPLE 3-14 `get_error_type()` Example

```
SEQUENCE {
    int INTEGER,
    char OCTET STRING
}
```

CODE EXAMPLE 3-14 `get_error_type()` Example (Continued)

```

    Result rslt = mbd.set("char", "10");
    if (rslt == NOT_OK) {
        ErrorType etype = mbd.get_error_type();
        cout << "Error string ==> " << mbd.get_error_string() <<
endl;
    }

```

The error type returned by the `get_error_type()` function can be any of the error types defined as `enum ErrorType` in `include/pmi/error.hh`.

Note – The Error type returned by the `MorfBuilder::set` function is the error type set by the `Morf::set` function since the `MorfBuilder::set` function is internally mapped to the `Morf::set` function.

The `ErrorType` from a function call must be accessed before a subsequent function call on the same `MorfBuilder` object because the error status from a previous function call is reset whenever a `MorfBuilder` function starts executing.

`get_error_string`

```
char* get_error_string(void) const;
```

Enables a `MorfBuilder` object to maintain the error status that results from the invocation of a `MorfBuilder` function until another function is invoked on the same `MorfBuilder`. The returned error string can be retrieved through the `get_error_string()` function.

For example, suppose a `MorfBuilder` object represents an object whose type is `SEQUENCE` (see CODE EXAMPLE 3-15).

CODE EXAMPLE 3-15 `get_error_string()` Example

```

SEQUENCE {
    int INTEGER,
    char OCTET STRING
}

Result rslt = mbd.set("char", "10");
if (rslt == NOT_OK) {

```

CODE EXAMPLE 3-15 `get_error_string()` Example (Continued)

```

        ErrorType etype = mbd.get_error_type();
        cout << "Error string ==> " << mbd.get_error_string() <<
endl;
    }

```

The error type returned by the `get_error_string()` function can be any of the error types defined as `enum ErrorType` in `include/pmi/error.hh`.

Note – The Error string returned by the `MorfBuilder::set` function is the error string set by the `Morf::set` function since the `MorfBuilder::set` function is internally mapped to the `Morf::set` function.

The error string from a function call must be accessed before a subsequent function call on the same `MorfBuilder` object because the error status from a previous function call is reset whenever a `MorfBuilder` function starts executing.

`set_prop`

```

Result set_prop (CDU key, CDU value);

```

Sets the property addressed by *key* to *value*. The properties are local to the `MorfBuilder` object and are used to control some of its behavior. TABLE 3-17 lists the currently defined properties for the `MorfBuilder` class. The `access_type` property is relevant only to a `MorfBuilder` class that represents a `SEQUENCE` type.

When the `access_type` property is set to `by_name`, all the `MorfBuilder` class functions that use the *navigation* string argument interpret the string as the member name of the `SEQUENCE` to be accessed.

When the `access_type` property is set to `by_index`, all the `MorfBuilder` functions that use the *navigation* string argument interpret the string as the position index for the member of the `SEQUENCE` to be accessed. CODE EXAMPLE 3-16 shows how to use the `set_prop()` function.

CODE EXAMPLE 3-16 `set_prop()` Example

```

Result rslt;
Morf morf;

rslt = mbd.set_prop("access_type", "by_name");

```

CODE EXAMPLE 3-16 `set_prop()` Example (Continued)

```
morf = mbd.get_raw("char");

rslt = mbd.set_prop("access_type", "by_index");
morf = mbd.get_raw("2");
```

```
Result set_prop (CDU navigation, CDU key, CDU value);
```

Invokes `set_prop(CDU key, CDU value)` on the sub-MorfBuilder member addressed by the navigation string.

3.20 PasswordTty Class

Inheritance: none

```
#include pmi/password_tty.hh
```

Data Members: No public data members are declared in this class.

This class implements the TTY based password query mechanism. You can derive from this class to implement different password query mechanism (for example, dialog box based for GUI applications). All non-GUI Solstice EM applications automatically get the TTY based password query mechanism. The `Platform::connect` method determines if user's password is required and accordingly calls a method on this class to get the password. If you don't wish to alter the default TTY based password query mechanism, you don't need to use this class.

At any given time, there can be at most one instance of this class or one of its derived classes. If no instance exists, then the `Platform::connect` method temporarily creates one and uses that to query the user's password. If you derive from this class and create an instance of it before you call `Platform::connect`, then that instance will be used to query the password. This is achieved using the virtual methods in C++. You need to implement the `PasswordTty::password_function` virtual method in your class to replace the default TTY based password query mechanism.

For an example program, please refer to files in `$EM_HOME/src/access_passwd` which demonstrate how to implement dialog box based password query mechanism in Motif based GUI applications.

3.20.1 Constructors

```
PasswordTty ()
```

Creates an instance of the `PasswordTty` class. This will result in an assertion failure if an instance of this or its derived classes already exists. At any given time, there can be at most one instance of this class or one of its derived classes.

3.20.2 PasswordTty Operator Overloading

No public operators are defined for this class.

3.20.3 PasswordTty Member function

```
password_function
```

```
virtual int password_function (char *user, char *password) const
```

This function queries the user for login name and password. The login name defaults to the `user` argument. This method returns a non-zero value if password query succeeds. If you derive from the `PasswordTty` class, you should implement this method to provide your own password query mechanism.

3.21 Platform Class

Inheritance: `public Error`

```
#include pmi/hi.hh
```

Data Members: No public data members are declared in this class.

An instance of the `Platform` class represents a potential or actual connection to a particular MIS, along with all the implied semantics of the framework implemented by the MIS.

The `Platform` is a reference-counting wrapper around an inner abstract base class; each framework derives a new class from the base class to implement framework-specific semantics. (The base class does provide a generic attribute-like mechanism that specific frameworks can use for specifying things like access tickets or default time-outs.).

TABLE 3-18 Platform Method Types

Method Name	Method Type
default_platform set_default_platform	MIS default
get_prop set_prop replace_discriminator replace_discriminator_classes	Property control
connect start_connect disconnect start_disconnect get_connection_fd cleanup_def_platform	MIS connection
find_album_by_nickname find_image_by_nickname find_image_by_objname find_image_by_oi get_fdn get_fullname get_shortcode	Name translation
get_when_syntax when	Function callback
get_plat_id get_raw_sap	Utility routines
get_attr_coder set_attr_coder	String encoding/decoding hooks
get_authorized_features get_authorized_applications	Access control

3.21.1 Constructors

```
Platform()
```

The default constructor creates a `Platform` instance that refers to no actual MIS. The value tests `FALSE` until you assign it a real `Platform` value.

```
Platform( const Platform& other)
```

The preceding is an ordinary copy constructor. After the copy, both copies still refer to the same MIS object. The reference count on the MIS object is incremented.

```
Platform( CDU platype, CDU nickname = duNO_VALUE() )
```

The preceding constructor constructs a `Platform` instance for a particular kind of MIS. Because a `Platform` is really a wrapper for a set of related classes, this function actually works a bit like a virtual constructor.

3.21.2 Destructor

```
~Platform()
```

Decrements the reference count and deletes the `Platform` if reference count is zero.

3.21.3 Platform Operator Overloading

```
Platform& operator = (const Platform& other)
```

The assignment operator, above, works like the copy constructor

```
operator void*()
```

The preceding cast operator is for use in conditionals. It returns `TRUE` if this `Platform` refers to an actual MIS object (regardless of whether that MIS is connected yet). Do not attempt to use the returned value as a pointer to anything, since it points to private data.

```
int operator !()
```

The preceding operator definition is provided so that you can state:

```
“if (!platform)...”
```

3.21.4 Platform Member Functions

This section describes the member functions of the `Platform` class.

`cleanup_def_platform`

```
static void cleanup_def_platform(void);
```

This method resets the default platform and clears the caches associated with the default platform. The `cleanup_def_platform` method should be invoked after disconnecting an existing platform and before attempting to establish a new platform connection.

The `cleanup_def_platform` method does not by itself force an established platform connection down. Rather it is used to clean up a connection after a disconnect (e.g. a disconnect event was received). The `cleanup_def_platform` method is usually used in the disconnect callback, registered for the platform.

Note – The attempt to reconnect to the platform must not occur in the disconnect callback. The disconnect callback only post a timer to invoke a reconnect callback sometimes later, after the disconnect callback has exited.

CODE EXAMPLE 3-17 is an example of this function's use:

CODE EXAMPLE 3-17 Platform::cleanup_def_platform Method

```

void disc_cb( Ptr, Ptr);
void mis_retry_handler (Ptr, Ptr);
Platform *plat = 0;
char *host;

int main (int argc, char **argv)
{
    host = getenv("EM_SERVER");
    if (!host) {
        host = new char[MAXHOSTNAMELEN+1];
        sysinfo(SI_HOSTNAME, host, 255);
    }
    while (1) {
        plat = new Platform(duEM);
        cout << "Connecting to ... " << host << endl;
        if (!plat->connect(host, "em_sample", 20.0)) {
            // Connect to the MIS
            cout << "Failed to connect to " << host << endl;
            cout << plat->get_error_string() << endl;
            Platform::cleanup_def_platform();
            delete plat;
            plat = 0;
        }
        else {
            break;
        }
    } /* while */

    if (!plat->when("DISCONNECTED", Callback(disc_cb, 0))) {
        cout << plat->get_error_string() << endl;
        exit(3);
    }

    .....

    while (TRUE) {
        dispatch_recursive(TRUE);
        // Enter the infinite listen loop.
    }
    exit(0);
}

.....

```

CODE EXAMPLE 3-17 Platform::cleanup_def_platform Method (*Continued*)

```

// Define a function to do something if disconnected.

void
disc_cb(Ptr, Ptr) {
    cout << "***** DISCONNECTED event received *****" << endl;
    Platform::cleanup_def_platform();
    delete plat;
    plat = 0;
    post_timer(Timer((10 * 1000), 0, mis_retry_handler, 0));
    return;
}
.....

void
mis_retry_handler (Ptr user_data, Ptr passed_data) {
    plat = new Platform(duEM);
    cout << "Connecting to ... " << host << endl;
    if (!plat->connect(host, "em_sample", 20.0)) {
        cout << "Failed to connect to " << host << endl;
        cout << plat->get_error_string() << endl;
        Platform::cleanup_def_platform();
        if (plat != 0) {
            delete plat;
            plat = 0;
        }
        post_timer(Timer((10 * 1000), 0, mis_retry_handler, 0));
        return;
    }
    if (!plat->when("DISCONNECTED", Callback(disc_cb, 0))) {
        cout << plat->get_error_string() << endl;
        exit(3);
    }
    return;
}

```

After a platform is disconnected, and the `cleanup_def_platform` is invoked, all the PMI objects used in the previous connection are invalid. New instances (Image and Album, for example) must be created and booted.

Note – Attempts to access PMI objects from the previous platform connection can result in segmentation violation, since the PMI program will attempt to access memory that has already been freed up. The PMI library does not support multiple concurrent platform connections within the same process (the same PMI program). After one platform is disconnected, the PMI program can invoke the `cleanup_def_platform` method to clear the platform caches and to reset the default platform, and then it can post a timer, that will invoke a reconnect callback and attempt to set up a new platform connection in that reconnect callback.

connect

```
Result connect(CDU location,
               CDU application_name,
               const Timeout to = DEFAULT_TIMEOUT)
```

This function call attempts to connect to the MIS. The value of *location* is implementation dependent, but might be something as simple as a host name.

This method determines if the user is subject to password authentication. If the user is, then this method queries the login name and password. By default, a tty-based password query mechanism is used. An application can redefine the password query mechanism by deriving from the `PasswordTty` class and providing its own definition of the `PasswordTty::password_function` virtual method.

The argument *application_name* is used by the Access Control Module to determine if the user has permission to use the application. If the user does not have permission to use the application this method fails to connect to the MIS and returns `NOT_OK`. The `Platform::get_error_string` function can be used to determine the reason for the failure.

default_platform

```
static Platform default_platform()
```

This function call returns the default MIS. The `Image` and `Album` constructors use this value if you invoke them without supplying an argument to specify an MIS. To set the default MIS, use `set_default_platform`.

disconnect

```
Result disconnect(const Timeout to = DEFAULT_TIMEOUT)
```

This method forces a disconnect from the platform, resets the default platform, and clears the caches associated with the default platform. Do not assume that the event callbacks are executed before the connection to the MIS is closed (so that it cannot be assumed that if a callback was registered for the DISCONNECTED event, that the callback will be invoked).

The invocation of `disconnect()` must happen in an event callback function, called indirectly through `dispatch_recursive()`. CODE EXAMPLE 3-18 sets a timer callback that will be invoked when the timer expires and will call `disconnect()`. A new timer can be set, within this callback, to execute a reconnect callback later.

CODE EXAMPLE 3-18 Platform::disconnect Method

```
void disc_cb( Ptr, Ptr);
void mis_retry_handler (Ptr, Ptr);
void disconnect_handler(Ptr, Ptr);

Platform *plat = 0;
char *host;

int main (int argc, char **argv)
{
    host = getenv("EM_SERVER");
    if (!host) {
        host = new char[MAXHOSTNAMELEN+1];
        sysinfo(SI_HOSTNAME, host, 255);
    }
    while (1) {
        plat = new Platform(duEM);
        cout << "Connecting to ... " << host << endl;
        if (!plat->connect(host, "em_sample", 20.0)) {
            // Connect to the MIS.
            cout << "Failed to connect to " << host << endl;
            cout << plat->get_error_string() << endl;
            Platform::cleanup_def_platform();
            delete plat;
            plat = 0;
        }
        else {
            break;
        }
    }
}
```


CODE EXAMPLE 3-18 Platform::disconnect Method (*Continued*)

```

        } /* while */

        if (!plat->when("DISCONNECTED", Callback(disc_cb, 0))) {
            cout << plat->get_error_string() << endl;
            exit(3);
        }

        post_timer(Timer((10 * 1000), 0, disconnect_handler, 0));

        .....

        while (TRUE) {
            dispatch_recursive(TRUE);
            // Enter the infinite listen loop.
        }
        exit(0);
    }

    .....
    // Define a function to do something if disconnected.

    void
    disc_cb(Ptr, Ptr) {
        cout << "***** DISCONNECTED event received *****" << endl;
        Platform::cleanup_def_platform();
        delete plat;
        plat = 0;
        post_timer(Timer((10 * 1000), 0, mis_retry_handler, 0));
        return;
    }

    .....

    void
    mis_retry_handler (Ptr    user_data,  Ptr passed_data) {
        plat = new Platform(duEM);
        cout << "Connecting to ... " << host << endl;
        if (!plat->connect(host, "em_sample", 20.0)) {
            cout << "Failed to connect to " << host << endl;
            cout << plat->get_error_string() << endl;
            Platform::cleanup_def_platform();
            if (plat != 0) {
                delete plat;
                plat = 0;
            }
        }
    }

```

CODE EXAMPLE 3-18 Platform::disconnect Method (*Continued*)

```

        post_timer(Timer((10 * 1000), 0, mis_retry_handler, 0));
        return;
    }

    if (!plat->when("DISCONNECTED", Callback(disc_cb, 0))) {
        cout << plat->get_error_string() << endl;
        exit(3);
    }
    return;
}

void
disconnect_handler(Ptr    user_data,  Ptr passed_data) {
    char ch;

    cout << " disconnecting !" << endl;
    if (plat) {
        plat->disconnect();
        delete plat;
        plat = 0;
        post_timer(Timer((10 * 1000), 0, mis_retry_handler, 0));
    }
}

```

After a platform is disconnected, and the `cleanup_def_platform` is invoked, all the PMI objects used in the previous connection are invalid. New instances (Image and Album, for example) must be created and booted.

Note – Attempts to access PMI objects from the previous platform connection can result in segmentation violation, since the PMI program will attempt to access memory that has already been freed up. The attempt to reconnect to the platform must not happen in the disconnect callback. The disconnect callback must only post a timer to invoke a reconnect callback sometimes later, after the disconnect callback has exited.

`find_album_by_nickname`

```
Album find_album_by_nickname( CDU name)
```

The preceding function call locates the album that has registered itself with the specified *name*. The null value `Album()` is returned if no such album is found.

`find_image_by_nickname`

```
Image find_image_by_nickname( CDU name)
```

The preceding function call locates the image that has registered itself with the specified *name*. The null value `Image()` is returned if no such image is found.

`find_image_by_objname`

```
Image find_image_by_objname( CDU name)
```

The preceding function call locates the image that has registered itself with the specified *name*. The null value `Image()` is returned if no such image is found. Note that the `Image` constructor can also do lookups for you, should you want the image to be created when not found

`find_image_by_oi`

```
Image find_image_by_oi( CDU oi)
```

This function returns the image that has registered itself with the specified *oi*. The null value `Image()` is returned if no such image is found.

`get_attr_coder`

```
Coder get_attr_coder( CDU attrname)
```

The preceding method returns the encoder/decoder for a given attribute.

`get_authorized_applications`

```
Result get_authorized_applications(AuthApps &apps,
const char *user = 0 );
```

This method is used to get the list of authorized applications for the given *user*. In most cases, you should use the default value of the *user* argument, which will default to the currently logged in user. The user logged in to MIS may be different than the user running the application since the user name can be changed during the password query.

This method should be called after `Platform::connect` has been successful. This method returns OK if there is no unexpected PMI error and if the user is authorized to use at least one application. You can query if an application is authorized or not by using the `AuthApps::is_authorized()` method on the *authApps* argument.

Please notice, this method uses the `AuthApps` class, whereas, the method `Platform::get_authorized_features` uses the `AuthFeatures` class. If all applications are authorized, any argument to the `AuthApps::is_authorized()` will return OK even if the application has not yet been registered in the MIS.

The following is an example of this function's use:

```
include pmi/auth_apps.hh
// After Platform::connect has been successful ...

AuthApps authApps;
if (platform.get_authorized_applications(authApps) == OK) {
    if (!authApps.is_authorized("em_discover")) {
        // em_discover is not authorized
    }
    // Check other apps ...
}
else {
    // handle unexpected PMI error
}
```

get_authorized_features

```
Result get_authorized_features (AuthFeatures &features,  
    const char *user = 0, const char *appname = 0 );
```

This method is used to get the list of authorized features for the given *user* and the *application*. In most cases, you should use the default value of the *user* and the *application* argument, which will default to the currently logged in user and the current application being run. The user logged in to MIS may be different than the user running the application since the user name can be changed during the password query.

This method should be called after `Platform::connect` has been successful. This method returns OK if there is no unexpected PMI error and if the user is authorized to use at least one feature for the given application. You can query if a feature is authorized or not by using the `AuthFeatures::is_authorized()` method on the *features* argument.

Please notice, this method uses the `AuthFeatures` class, whereas, the method `Platform::get_authorized_applications` uses the `AuthApps` class. If all features are authorized, any argument to the `AuthFeatures::is_authorized()` will return OK even if the feature has not yet been registered in the MIS.

Platform Class

The following is an example of this function's use:

```
#include pmi/auth_features.hh
// After Platform::connect has been successful ...

AuthFeatures features;
if (platform.get_authorized_features(features) == OK) {
    // For the sake of this example, it is assumed,
    // there are two features, "Create" and "Delete", in
    // this application. It is also assumed that by default
these
    // features are enabled. The following code
    // checks if a feature is not authorized and disables
the feature.

    if (!features.is_authorized("Create")) {
        // Disable "Create" feature
    }
    if (!features.is_authorized("Delete")) {
        // Disable "Delete" feature
    }
}
else {
    // handle unexpected PMI error
}
```

get_connection

```
int get_connection_fd()
```

The preceding method returns the file descriptor corresponding to the connection to an MIS. It is useful in a GUI program for waiting in a `select()` (3C) for platform events.

get_fdn

```
DU get_fdn (DU &dn)
```

The preceding method returns the *ldn* or *fdn* to the specified *dn*.

Platform Class

get_fullname

```
DU get_fullname( CDU shortname )
```

The preceding function call translates a short attribute name to its fully qualified name. Refer also to the `Platform::when` method description for more information.

get_plat_id

```
PlatformId get_plat_id()
```

The preceding function call returns the MIS ID number, which is not very useful outside the PMI.

get_prop

```
DU get_prop( CDU key )
```

The preceding function call returns a property of the current MIS. If *key* does not specify an existing property, returns a null `DataUnit`, which tests `FALSE` in a conditional. Most MISs support, at a minimum, the properties in TABLE 3-19.

TABLE 3-19 MIS Properties Supported

Properties	Description
PLATFORM_TYPE	As specified to the “virtual” constructor. (Read Only)
PLATFORM_OBJNAME	Absolute object name of the MIS itself. (Read Only)
APPLICATION_OBJNAME	Object instance name of the object (within the MIS) that represents the current application's connection and/or process. (Read Only)
PLATFORM_NICKNAME	The nickname for the MIS that you specified to the “virtual” constructor.
APPLICATION_TYPE	The kind of application specified to <code>connect</code> . (Read Only)

TABLE 3-19 MIS Properties Supported (*Continued*)

Properties	Description
LOCATION	The location of the MIS specified to connect. (Read Only)
STATE	The state of the connection (Read Only): DOWN - Unconnected BOOT - Connecting UP - Connected SHUTDOWN - Disconnecting
DEFAULT_TIMEOUT	The default time-out for this MIS, in seconds, with fractions allowed.

If you want to construct an event sieve by hand, you need to know the application instance name to tell the sieve where to forward events. The PMI constructs most sieves for you automatically, so you generally need not be concerned with this.

The PMI makes no use of the nickname property; it is there merely as a convenience.

The synchronous version of any function that has both a synchronous and an asynchronous version uses the default time-out. If you set the default time-out to 0, only the asynchronous version works (unless of course you specify an explicit time-out on the synchronous call itself).

See the description of the `replace_discriminator()` and `replace_discriminator_classes()` methods, below. These methods allow you, to construct an event seive, without a call to `get_prop()` in some programs.

replace_discriminator

Result `replace_discriminator` (DU *discriminatorConstruct*)

The preceeding function call takes a discriminator construct, for example, (and: {item:equality...}). In certain cases using this function can allow you to avoid a call to `get_prop()`. Using this function reduces network traffic and unnecessary processing time. This function is more generalized and more difficult to use than the `replace_discriminator_classes` function.

`replace_discriminator_classes`

```
Result replace_discriminator_classes (Array (DU) object_classes,
                                     Array (DU) event_types = DEF_ARRAY_DU)
```

The preceding function call takes an array of managed object classes, such as the topo managed object class, and an array of events, such as object create. This function works on an application instance object. In certain cases, using this function can allow you to avoid a call to `get_prop()`. Using this function reduces network traffic and unnecessary processing time.

`get_raw_sap`

```
void* get_raw_sap()
```

The preceding function call returns the message SAP of the internal connection to the MIS. This is useful for programs that need to use both the low- and high-levels of the PMI, when both need to share the same connection.

`get_shortcode`

```
DU get_shortcode( CDU fullname)
```

The preceding function call translates a fully qualified attribute name to its corresponding short name. Typically this involves stripping a document name from it.

`get_when_syntax`

```
Syntax get_when_syntax( CDU eventname)
```

The preceding function call returns the syntax of the information that is passed to the callback function. Primarily for internal use.

Platform Class

set_attr_coder

```
void set_attr_coder( CDU attrname, Coder cd)
```

The preceding function call sets the encoder/decoder for a given attribute. The Coder is used by the `get_str` and `set_str` functions in both `Morfs` and `Images`. See also `set_coder`, under the description of the `Syntax` class.

set_default_platform

```
static void set_default_platform( Platform& plat)
```

The preceding function call sets the default MIS. The `Image` and `Album` constructors use this value when you omit the `platform` argument. You do not usually need to call this function, since the first time you connect to a MIS it is invoked for you, and most applications talk to only one MIS.

set_prop

```
Result set_prop( CDU key, CDU value)
```

The preceding function call sets a property of the current MIS, provided that the *key* specifies a supported property and the *value* specifies a legal value. If `set_prop` cannot do what you ask, it throws an invalid exception. Refer to TABLE 3-19 for some typical properties.

start_connect

```
Waiter start_connect( CDU platform,  
                     CDU application_name,  
                     CCB cb = NO_CALLBACK)
```

The preceding function call is the asynchronous version of `connect`.

Platform Class

start_disconnect

```
Waiter start_disconnect( CCB cb = NO_CALLBACK)
```

The preceding function call is the asynchronous version of `disconnect`.

when

```
Result when( CDU eventname, CCB cb = NO_CALLBACK)
```

The Platform object receives all events at which time all callbacks registered for by the Platform objects are executed. Next, all of the callbacks registered for by image objects are executed, then all of the callbacks registered for by album objects are executed.

The name of the event specified in the above call needs to be the fully specified name defined in the GDMO definition unless it is a standard event supported by the Platform such as `OBJECT_CREATED`, `ATTR_CHANGED` etc. Refer also to the `Platform::get_fullname` method description.

The following function call establishes a callback routine to handle a class of MIS-specific asynchronous events.

For example, the following could be used to find out when an MIS disconnects:

```
when( "DISCONNECTED", Callback( disconnect_cb, 0));
```

TABLE 3-20 shows the supported Platform events.

TABLE 3-20 MIS Events Supported

Event	Description
ATTR_CHANGED	An attribute of an object in the MIS changed.
DISCONNECTED	The MIS dropped its end of the connection for some reason.
OBJECT_CREATED	An object was created in the MIS.

TABLE 3-20 MIS Events Supported (*Continued*)

Event	Description
OBJECT_DESTROYED	An object was destroyed in the MIS.
RAW_EVENT	A raw event came in from the MIS. You can examine it before the PMI does anything else with it. But note that <code>CurrentEvent::do_something</code> never does anything with a raw event. You have to register a callback for the event under the proper name, such as <code>ATTR_CHANGED</code> , and <code>do_something</code> in that callback.
WAIT	The PMI is entering a wait state. <code>CurrentEvent::get_name</code> returns a string describing the wait state. The callback is called again when leaving the wait state, but with a null name.

3.21.5 GETENV Macro

```
#include pmi/installation.hh
GETENV(key);
```

The GETENV macro returns a null terminated string with the value of the environment variable pointed to by *key*, or NULL when there is no such environment variable. The parameter *key* is of type `char *`.

3.22 Syntax Class

Inheritance: `public Error`

```
#include pmi/hi.hh
```

Data Members: No public data members are declared in this class.

A `Syntax` is the representation of a type. All framework-encoded data, whether part of an object or not, has a type. This type specifies, among other things, how to produce and understand human-readable representations of the data. In general, the application programmer need not deal with `Syntaxes` directly except when building `Morfs` from scratch.

TABLE 3-21 Syntax Method Types

Method Name	Method Type
expansion get get_raw get_type	Access to the type representation
get_platform is_any is_choice is_list is_sequence is_set	Collateral information
get_member_names get_memname get_members member	List handling
get_coder set_coder	String encoding/decoding hooks

3.22.1 Constructors

```
Syntax( )
```

The default constructor creates a `Syntax` instance that refers to no actual `Syntax`. The value tests `FALSE` until you assign it a real `Syntax` value.

```
Syntax( const Syntax& other)
```

The preceding constructor is an ordinary copy constructor. After the copy, both copies still refer to the same `Syntax` object. The reference count on the `Syntax` object is incremented.

```
Syntax( Platform& plat, DU text)
```

The preceding constructor constructs a `Syntax` instance for a particular kind of MIS. Because a `Syntax` is really a wrapper for a set of related classes, this function actually works somewhat like a virtual constructor.

```
Syntax( Morf& morf)
```

The preceding constructor constructs a `Syntax` instance for a particular kind of MIS, which is an implicit part of the `Morf`. The data portion of the `Morf` is interpreted as a description of the correct `Syntax`. (The `Syntax` implicit to the `Morf` describes the syntax of the description, not the `Syntax` to be created by the constructor.) Because a `Syntax` is really a wrapper for a set of related classes, this function actually works a little like a virtual constructor.

```
Syntax(CDU attrname, Platform& plat = Platform::def_platform)
```

The preceding constructor constructs a `Syntax` instance for a particular kind of attribute. You could conceivably accomplish the same end by creating an image for an object of the type containing the attribute in question, extracting the `Morf` corresponding to that attribute, and then extracting the `Syntax` from that `Morf`. However, this is far more cumbersome than simply instantiating a `Syntax` instance.

3.22.2 Syntax Operator Overloading

```
Syntax& operator = (const Syntax& other)
```

The assignment operator works like the copy constructor.

```
operator void*()
```

The preceding cast operator is for use in conditionals. It returns `TRUE` (that is, a nonzero value) if this `Syntax` refers to an actual syntax object. Do not attempt to use the returned value as a pointer to anything, since it points to private data.

```
int operator !()
```

The preceding function call is provided so that you can use “if (!syntax)...”.

3.22.3 Syntax Member Functions

This section describes the member functions of the `Syntax` class.

`expansion`

```
Syntax expansion()
```

The purpose of the `Syntax::expansion()` is to return the syntax, without respect to tagging or selection, from the current syntax.

`get`

```
DU get(FBits fb = 0)
```

Returns a textual expansion of the syntax

`get_coder`

```
Coder get_coder()
```

The preceding function call returns the encoder/decoder for this type.

`get_member_names`

```
Array(DU) get_member_names()
```

The preceding function call returns the list of member names (field names) for list types that have such names.

Syntax Class

get_members

```
Array(syntax) get_members()
```

The preceding function call returns a list of syntax for the corresponding member name (field name).

get_memname

```
DU get_memname()
```

The preceding function call returns the member name (field name) of this syntax, if it happens to be a member of a list that has member names.

get_platform

```
Platform get_platform()
```

The preceding function call returns the MIS implicit in every *Syntax*.

get_raw

```
Morf get_raw()
```

The preceding function call returns the *Syntax* in encoded, MIS-specific form.

Syntax Class

get_type()

```
Asn1Type get_type()
```

This method returns the underlying `Asn1Type` object associated with a `Syntax` object. If the `Syntax` object is not initialized, the method returns a `NULL Asn1Type` (CODE EXAMPLE 3-19).

CODE EXAMPLE 3-19 Syntax::get_type() Example

```
Syntax tstsyn = tstmrf.get_syntax();
Asn1Type tsttype2 = tstsyn.get_type();
if (tsttype2) {
    cout << "The Asn1Type is initialized!" << endl;
} else {
    cout << "The Asn1Type is not initialized!" << endl;
}
```

is_any

```
Boolean is_any()
```

The preceding function call returns `TRUE` if the syntax describes a type that is any.

is_choice

```
Boolean is_choice()
```

The preceding function call returns `TRUE` if the `Syntax` describes a type that lets you pick one of a set of other types.

Syntax Class

is_list

```
Boolean is_list()
```

The preceding function call returns TRUE if the `Syntax` describes a compound data value, and FALSE if it describes a scalar value.

is_sequence

```
Boolean is_sequence()
```

The preceding function call returns TRUE if the `syntax` describes a compound data value that contains an ordered list of zero or more members.

is_set

```
Boolean is_set()
```

The preceding function call returns TRUE if the `syntax` describes a compound data value that contains an unordered list of zero or more members.

member

```
Syntax member( DU name = duNO_VALUE )
```

The preceding function call returns the `Syntax` for a type that is a member of the current type. Obviously, this is valid only for `Syntaxes` that have members, namely choices and lists. If a list type has unnamed members (or is a list of identical elements, such as a “SET OF” or “SEQUENCE OF” ASN-1 type) the first anonymous member type can be extracted by passing a null *name* argument. (Subsequent anonymous types are inaccessible.)

Waiter Class

set_coder

```
void set_coder(  Coder  coder)
```

The preceding function call sets the encoder/decoder for this type. The *Coder* is used by the *get_str* and *set_str* functions in both *Morfs* and *Images*. See also *set_attr_coder* under the description of the *Platform* class.



3.23 Waiter Class

Inheritance: public *Error*

```
#include pmi/hi.hh
```

Data Members: No public data members are declared in this class.

A *Waiter* is the representation of an ongoing asynchronous operation. The *Waiter* provides methods for cancelling and awaiting completion of the operation.

The *Waiter* can also serve as the basis for asynchronous operations of your own construction.

TABLE 3-22 Waiter Method Types

Method Name	Method Type
wait	Normal Wait

TABLE 3-22 Waiter Method Types *(Continued)*

Method Name	Method Type
when_canceled when_done when_resp when_tick	Schedule additional notifications
cancel get_except num_clobbered time_remaining waitmore was_completed	Managing a Waiter you didn't create
clobber complete dec get_current_event get_data inc send_resp ref	Managing a Waiter you created

3.23.1 Constructors

```
Waiter()
```

The default constructor creates a `Waiter` instance that refers to no actual `Waiter`. The value tests `FALSE` until you assign it a real `Waiter` value.

```
Waiter( const Waiter& other)
```

The preceding constructor is an ordinary copy constructor. After the copy, both copies still refer to the same `Waiter` object. The reference count on the `Waiter` object is incremented.

```
Waiter( Ptr ptrdata,  
CCB callback,  
Timeout defto = REAL_DEFAULT_TIMEOUT )
```

Waiter Class

The preceding constructor constructs a `Waiter` instance that calls back to the specified *callback* when the `Waiter` completes. The *ptrdata* is a convenient place to store arbitrary data.

```
Waiter( Ptr ptrdata, Boolean reuse = FALSE )
```

The preceding constructor constructs a `Waiter` instance from a `void*` pointer created by the `ref` method. It is primarily for internal PMI use within callbacks, when the callback can occur after the original `Waiter` has gone out of scope, and would ordinarily have been deleted. Each call to `ref` increments a reference count, and each construction of a `Waiter` using this constructor eventually causes the reference count to be decremented again when the `Waiter` is destructed at the end of the callback. If multiple callbacks are to use the same pointer, then pass a *reuse* parameter of `TRUE` on all but the last callback (or call `ref` again within the callback) to keep the reference alive till the next callback.

```
Waiter::complete
```

After the preceding constructor is constructed from a `void*` pointer by the `ref()` method, make sure the `waiter::complete()` function is called. An example is shown below.

CODE EXAMPLE 3-20 waiter::complete() Function

```
Callback cb_all(all_done, 0);
Waiter all_waiter(0, cb_all);

cout << "Image(" << obj1 << ") to be booted" << endl;
Image im1 = Image(obj1);
Callback cb1(f1, all_waiter.ref());
Waiter waiter1 = im1.start_boot(cb1);

cout << "Image(" << obj2 << ") to be booted" << endl;
Image im2 = Image(obj2);
Callback cb2(f2, all_waiter.ref());
Waiter waiter2 = im2.start_boot(cb2);

cout << "Waiting for callback event" << endl;

while ( !done_flag ) {
    dispatch_recursive(TRUE);
}
```

CODE EXAMPLE 3-20 waiter::complete() Function (Continued)

```
void
f1(Ptr user_data, Ptr)
{
    cout << "job1 is done" << endl;

    Waiter waiter1(user_data, FALSE);

    if (waiter1)
    {
        //Mark job1 is done
        waiter1.complete();
    }
}

void
f2(Ptr user_data, Ptr)
{
    cout << "job2 is done" << endl;

    Waiter waiter2(user_data, FALSE);

    if (waiter2)
    {
        //Mark job2 is done
        waiter2.complete();
    }
}

void
all_done(Ptr, Ptr)
{
    cout << "All Jobs are done" << endl;

    done_flag = 1;
}
```

3.23.2 Waiter Operator Overloading

```
Waiter& operator = ( const Waiter& other)
```

The assignment operator, above, works like the copy constructor.

```
operator void*()
```

The preceding cast operator is for use in conditionals. It returns TRUE if this `Waiter` refers to an actual `Waiter` object. Do not attempt to use the returned value as a pointer to anything, since it points to private data.

```
int operator !()
```

The preceding function call is provided so that you can say
“if (!waiter)...”

3.23.3 Waiter Member Functions

This section describes the member functions of the `Waiter` class.

`cancel`

```
Result cancel()
```

The preceding function call causes the asynchronous operation to time out immediately. Essentially it's a `waitmore(0.0)`. Any existing callbacks are not removed.

Waiter Class

clobber

```
void clobber( const ExceptionType* err = 0)
```

The preceding function call causes the `Waiter` to be marked as deficient in some respect or other. This function is primarily for internal use; the PMI uses it to pass error information back to the application when, for instance, an unexpected error response is received from the MIS. The function can be called multiple times, but only the first error is remembered. This function does not complete the `Waiter`.

complete

```
void complete()
```

The preceding function call marks the `Waiter` as complete. This function is primarily for internal use; the PMI uses it to notify the `Waiter` so that it can call any waiting external callbacks, and can let the `wait` function return to the application (if the `wait` function was in fact called). `Waiter.complete()` should be called regularly. Calls to this function are ignored if there are still pending internal callbacks.

dec

```
U32 dec()
```

The preceding function call decrements the `Waiter`'s count of the number of internal callbacks it is waiting for. The `Waiter` cannot complete while there are pending internal callbacks. This function is primarily for internal use; it returns the number of callbacks pending after the decrement.

Waiter Class

get_current_event

```
CurrentEvent get_current_event()
```

The preceding function call returns the `CurrentEvent` that is contained within the `Waiter`. This is the event that is passed to all external callbacks when the `Waiter` completes. The “message pointer” in this event is actually a pointer back to the `Waiter` containing it.

get_data

```
Ptr get_data()
```

The preceding function call returns the *ptrdata* originally passed to the `Waiter` constructor. This function is primarily for internal use.

get_except

```
const ExceptionType* get_except()
```

The preceding function call returns the exception that the `Waiter` was clobbered with, if any.

inc

```
U32 inc()
```

The preceding function call increments the `Waiter`’s count of the number of internal callbacks it is waiting for. The `Waiter` cannot complete while there are still pending internal callbacks. This function is primarily for internal use. It returns the number pending before the increment.

Note – It can be disastrous to lose track of the number of pending callbacks. You can either hang forever or dump core.

Waiter Class

num_clobbered

```
U32 num_clobbered()
```

The preceding function call returns the number of times the `Waiter` was clobbered.

ref

```
Ptr ref()
```

The preceding function call returns a reference-counted (`refcnt`) `void*` pointer to this `Waiter`, from which you *must*, at some future time, reconstruct the `Waiter` using the `Waiter(Ptr, Boolean)` constructor. Refer to the description of that constructor earlier in this section for further information.

Note – If `Waiter::ref()` is used to pass waiters in callbacks, then the asynchronous operation never completes. This is because the `ref()` function may do more than increase the ref count on the `Waiter`. It also increases the pending count on the `Waiter`.

send_resp

```
void send_resp(Ptr Calldata)
```

When invoked on a `Waiter`, this method posts the callback, stored during a previous call to the `when_resp()` method, in the scheduler queue with the associated *Calldata*. *Calldata* is a pointer that can be converted to a `CurrentEvent` object which can be manipulated by the application that receives it.

Waiter Class

time_remaining

```
Timeout time_remaining()
```

The preceding function call returns the time remaining before the `Waiter` would expire due to a time-out. In combination with the `when_tick` function, this lets you give the user a countdown till the time the application blows up. Note that a `Timeout` value is a (possibly fractional) number of seconds. Any rounding is up to you.

wait

```
CurrentEvent wait( const Timeout to = DEFAULT_TIMEOUT)
```

The preceding function call blocks for up to the specified period, waiting for the asynchronous operation to complete. It returns a `CurrentEvent`. It is also capable of throwing an exception if the `Waiter` was clobbered with a nonzero `ExceptionType` pointer.

waitmore

```
Result waitmore( const Timeout to = DEFAULT_TIMEOUT)
```

The preceding function call is available for callbacks to reset the time-out clock to a longer (or shorter) interval because some intermediate event occurred. For instance, if receiving multiple messages, you might want to extend the time-out each time a message comes in, and only time out if the gap between two subsequent messages exceeds some threshold.

was_completed

```
Boolean was_completed()
```

The preceding function call returns `TRUE` if the `Waiter::complete` function has been called when there were no more pending internal callbacks.

Waiter Class

when_canceled

```
Result when_canceled( CCB cb = NO_CALLBACK )
```

The preceding function call specifies a callback to call if the operation is cancelled or times out or is otherwise clobbered. Multiple callbacks can be added per waiter.

Note – All of the callbacks will be called if multiple callbacks are issued on the `Waiter`. Multiple callbacks are supported so that more than one area of the program code can be notified of the completion of an asynchronous operation.

If the callback is not specified, all callbacks are removed, *including* the original callback passed to the constructor.

when_done

```
Result when_done( CCB cb = NO_CALLBACK )
```

The preceding function call specifies a callback to call if the operation completes successfully. Multiple callbacks can be added per waiter.

Note – All of the callbacks will be called if multiple callbacks are issued on the `Waiter`. Multiple callbacks are supported so that more than one area of the program code can be notified of the completion of an asynchronous operation.

If the callback is not specified, all callbacks are removed, *including* the original callback passed to the constructor.

when_resp

```
Result when_resp(CCB cb = NO_CALLBACK )
```

When invoked on a `Waiter` returned from one of the `Album` methods, this function allows the given callback *cb* to be called each time a reply related to the sent request is available.

The callback's second argument, which is a pointer of type `void`, cannot be ignored and the callback must build a `CurrentEvent` with the second argument.

Waiter Class

CODE EXAMPLE 3-21 is an example of appropriate callback use.

CODE EXAMPLE 3-21 Appropriate Callback Use Example

```
void cb(Ptr userdata, Ptr calldata)
{
    // do whatever
    if(calldata)
    {
        CurrentEvent ce(calldata);
        // Do whatever and use and access the information
        // within the CurrentEvent ce
    }
    // do whatever
}
```

CODE EXAMPLE 3-22 is an example of inappropriate callback use.

CODE EXAMPLE 3-22 Inappropriate Callback Use Example

```
void cb(Ptr userdata,Ptr calldata)
{
    // do whatever but never use calldata to

}
// or
void cb(Ptr userdata)
{
    // do whatever and ignore the second argument
}
```

The relevant information in the `CurrentEvent` depends on the request sent in the case of `CMIS M_GET`, `M_SET`, or `M_DELETE`.

The following methods are available on the `CurrentEvent` object instance:

- `get_objname()`
- `get_objclass()`
- `get_image()`
- `get_album()`
- `get_message()`

Waiter Class

The `get_message()` method returns the message that caused the callback function to be called. And in the case of `CMIS M_ACTION`, two additional methods are added to the `CurrentEvent` object instance:

- `get_eventtype()`: Returns the name of the action.
- `get_info_raw()`: Returns a `Morf` instance that contains the `action_reply` data member of the message.

`when_tick`

```
Result when_tick( CCB cb = NO_CALLBACK, const Timeout to = 1.0)
```

The preceding function call specifies a callback to call repeatedly as long as the operation has not yet completed successfully. By default the callback is called once per second, but you can specify a different `Timeout` parameter to modify that. Only one such callback can be added per waiter. If the callback is not specified, the callback is removed.

Low-Level PMI

The Solstice EM Portable Management Interface (PMI) provides a low-level Common Management Information Server (CMIS)-like, distributed, transport-independent interface for application programs. CMIS-like messages are sent and received across this interface. These messages are routed to and from the Message Routing Module (MRM) component of the Management Information Server (MIS).

This chapter comprises the following topics:

- Section 4.1 “Communication Path” on page 4-1
- Section 4.2 “Root Classes for the Low-Level PMI” on page 4-3
- Section 4.3 “Low-Level PMI Classes” on page 4-3
- Section 4.69 “Constants and Defined Types” on page 4-81

4.1 Communication Path

The Low-Level PMI uses paired sets of Transport-Independent and Transport-Dependent Service Access Points (SAPs) to provide a communication path between an application and the MRM. The set of paired SAPs use the transport mechanism specified by the transport dependent SAP to pass messages between SAPs (between an application and the MRM). Each set of SAPs normally resides in separate Unix processes. Solstice EM currently uses a CMIS-like protocol over the Lightweight Presentation Protocol (LPP) and TCP/IP to pass messages between the paired SAPs.

Communication Path

FIGURE 4-1 shows how the Low-Level PMI communicates between applications and the MRM.

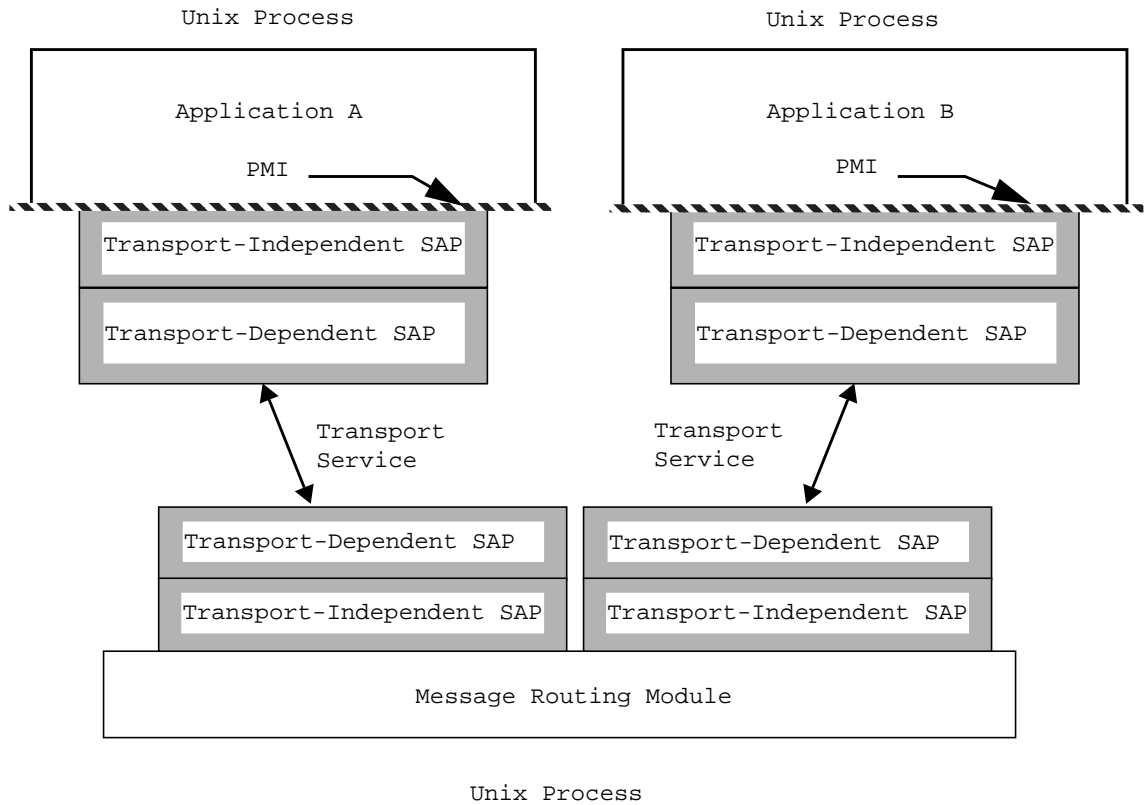


FIGURE 4-1 Applications to MRM Communication

4.2 Root Classes for the Low-Level PMI

The root classes for the Low-Level PMI include:

- Message Class
- MessageSAP Class
- MessScope Class

In addition, the `Asn1Value` and `DataUnit` classes are common base classes.

The data contained in the class structures based on `Message` class and defined in the `/opt/SUNWconn/em/include/pmi/message.hh` file are primarily based on the `Asn1Value` class, defined in `asn1_val.hh`. The `Asn1Value` class in turn relies on structures and methods defined in the `DataUnit` class, defined in `du.hh`.

4.3 Low-Level PMI Classes

4.3.1 Class Inheritance

The `Message` class is the base class used by almost all messages passed between SAPs and the PMI. The messages contained in the `message.hh` file largely define the syntax of the low-level usage of the PMI.

Low-Level PMI Classes

A number of CMIS-like messages are subclasses of the `Message` class. The three primary types of messages are:

- Request messages
- Response messages
- Error response messages

Solstice EM CMIS messages are derived from `Message`, the base message class. It contains data that is common to every type of message sent via a `MessageSAP` interface. All of the other base message classes derive either directly or indirectly from the `Message` class. You should never instantiate this class, only derive other classes from it.

Other base message classes contain a parameter or set of parameters that are commonly used together in more than one Solstice EM CMIS message. Some messages commonly use more than one set of parameters and therefore, some of the base message classes are combinations of other base message classes formed via inheritance. The parameters that are included in the base message classes include only those parameters that are used by the Message Routing Module (MRM) to perform scoping, filtering, access control, and synchronization.

Note – Each of the classes derived from the `Message` class relies on the ISO specifications of the CMIP protocol and ASN.1 data encoding.

FIGURE 4-2 shows the inheritance hierarchy for the classes based on Message:

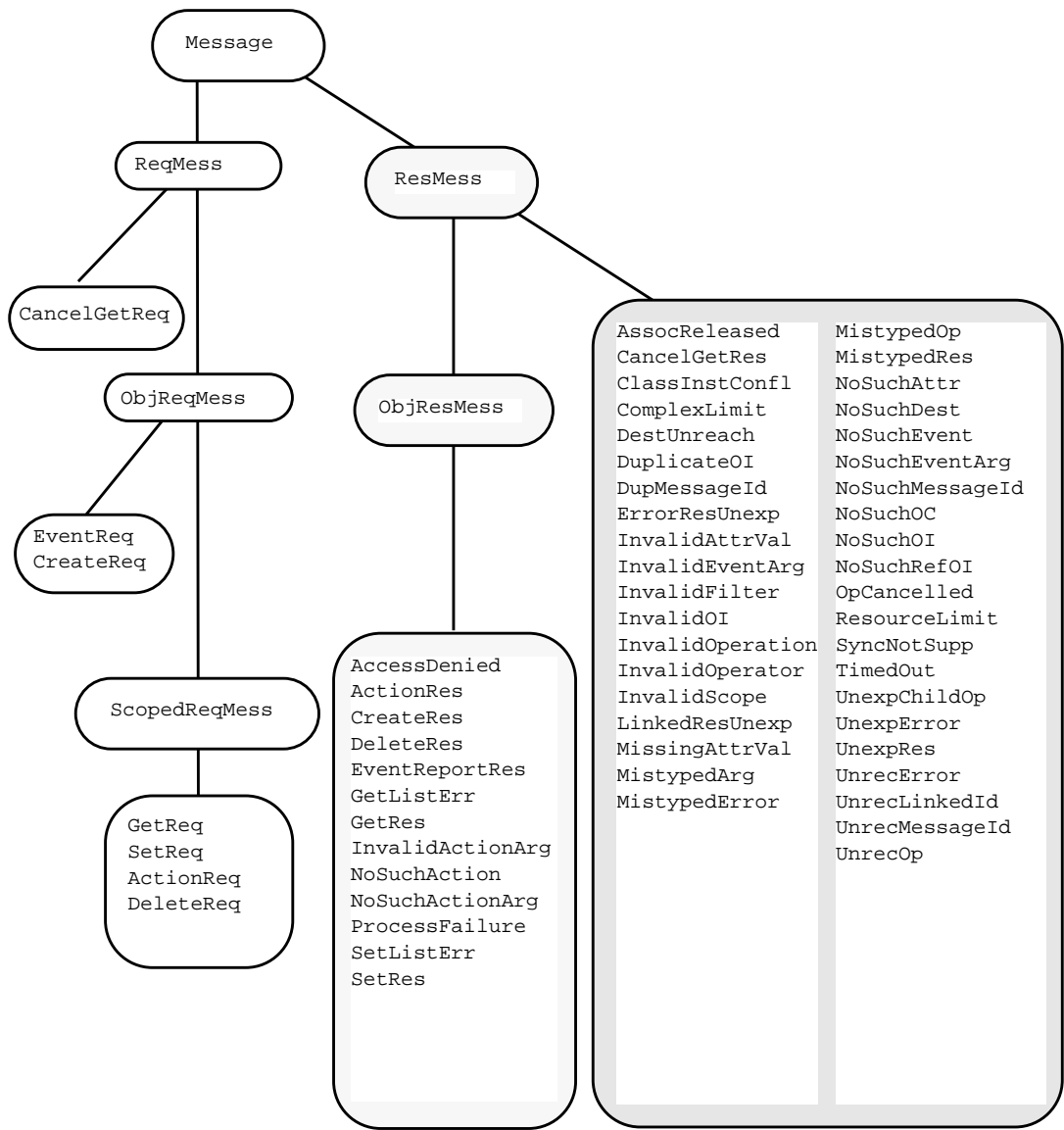


FIGURE 4-2 Inheritance Tree of the Message Class

4.3.2 Class Summary

TABLE 4-1 lists the Low-Level PMI classes.

TABLE 4-1 Low-Level PMI Classes

Class	Description
AccessDenied	Encapsulates an AccessDenied error
ActionReq	Serves as a repository for information identifying an action request message
ActionRes	Represents successful completion and results of an action request
AssocReleased	Represents an AssocReleased message
CancelGetReq	Encapsulates a CancelGet message to stop a lengthy Get request
CancelGetRes	Response to a CancelGet message
ClassInstConfl	Encapsulates a ClassConflict message
CreateReq	Serves as a repository for information identifying a create request message
CreateRes	Represents successful completion and results of a create request
DeleteReq	Serves as a repository for information identifying a delete request message
DeleteRes	Represents successful completion and results of a delete request
DuplicateOI	Encapsulates a DuplicateOI error
DupMessageId	Encapsulates a DupMessageId error
ErrorResUnexp	Encapsulates an ErrorResUnexp error
EventReq	Adds parameters to store an event type, time and information
GetListErr	Encapsulates a GetList error
GetReq	Serves as a repository for information identifying a get request message
GetRes	Represents successful completion and results of a get request
InvalidActionArg	Encapsulates an InvalidActionArg error
InvalidAttrVal	Encapsulates an InvalidAttrVal error

TABLE 4-1 Low-Level PMI Classes (*Continued*)

Class	Description
InvalidEventArgs	Encapsulates an InvalidEventArgs error
InvalidFilter	Encapsulates an InvalidFilter error
InvalidOI	Encapsulates an InvalidOI error
InvalidOperation	Encapsulates an InvalidOperation error
InvalidOperator	Encapsulates an InvalidOperator error
InvalidScope	Encapsulates an InvalidScope error
LinkedResUnexp	Encapsulates a LinkedResUnexp error
Message	Contains data that is common to every type of message sent via a MessageSAP interface
MessageSAP	Defines queues of pointers to messages
MessQOS	Represents the Quality of Service indicator included in all messages
MessScope	Defines a message's scope—the range of objects where a message is applied
MissingAttrVal	Encapsulates a MissingAttrVal error
MistypedArg	Encapsulates a MistypedArg error
MistypedError	Encapsulates a MistypedError error
MistypedOp	Encapsulates a MistypedOp error
MistypedRes	Encapsulates a MistypedRes error
NoSuchAction	Encapsulates a NoSuchAction error
NoSuchActionArg	Encapsulates a NoSuchActionArg error
NoSuchAttr	Encapsulates a NoSuchAttr error
NoSuchEvent	Encapsulates a NoSuchEvent error
NoSuchEventArgs	Encapsulates a NoSuchEventArgs error
NoSuchMessageId	Encapsulates a NoSuchMessageId error
NoSuchOC	Encapsulates a NoSuchOC error
NoSuchOI	Encapsulates a NoSuchOI error
NoSuchRefOI	Encapsulates a NoSuchRefOI error
ObjReqMess	Encapsulates an ObjReqMess message
ObjResMess	Encapsulates an ObjResMess message
OpCancelled	Encapsulates an OpCancelled message

Low-Level PMI Classes

TABLE 4-1 Low-Level PMI Classes *(Continued)*

Class	Description
ProcessFailure	Encapsulates a ProcessFailure error
ReqMess	Encapsulates a ReqMess message
ResMess	Encapsulates a ResMess message
ResourceLimit	Encapsulates a ResourceLimit message
ScopedReqMess	Encapsulates a ScopedReqMess message
SetListErr	Encapsulates a SetListErr error
SetReq	Encapsulates a SetReq message
SetRes	Encapsulates a SetRes message
SyncNotSupp	Encapsulates a SyncNotSupp message
TimedOut	Encapsulates a TimedOut error
UnexpChildOp	Encapsulates an UnexpChildOp error
UnexpError	Encapsulates an UnexpError error
UnexpRes	Encapsulates an UnexpRes error
UnrecError	Encapsulates an UnrecError error
UnrecLinkId	Encapsulates an UnrecLinkId error
UnrecMessageId	Encapsulates an UnrecMessageId error
UnrecOp	Encapsulates an UnrecOp error

4.4 AccessDenied Class

Inheritance: public ObjResMess, public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the `AccessDenied` class adds an `AsnlValue` parameter to store a current time. The `oc`, `oi`, and `curr_time` members are only defined when returning an error from a scoped `ACTION_REQ` or `DELETE_REQ`.

TABLE 4-2 lists the `AccessDenied` public data member.

TABLE 4-2 AccessDenied Public Data Member

Type	Variable	Description
AsnlValue	curr_time	An optional parameter specifying the time that this response message was generated.

4.4.1 Constructor

```
AccessDenied()
```

The constructor for `AccessDenied` takes no parameters. It initializes its parent class.

4.5 ActionReq Class

```
Inheritance:public ScopedReqMess, public ObjReqMess, public  
ReqMess, public Message  
  
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

An instance of `ActionReq` serves as a repository for information identifying an action request message.

TABLE 4-3 lists the `ActionReq` public data members.

TABLE 4-3 ActionReq Public Data Members

Type	Variables	Description
Asn1Value	action_type	The type of action being requested by this message
Asn1Value	action_info	Information that might be included in this action request. The data content of this parameter depends on the <code>action_type</code> . There are definitions for the action types in the OSI Network Management Forum document.

4.5.1 Constructor

ActionReq()

The constructor for `ActionReq` takes no parameters and does nothing more than initialize its parent classes.

4.6 ActionRes Class

Inheritance: public ObjResMess, public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the `ActionRes` class adds three `AsnlValue` parameters to store a current time, an action type, and some action reply information.

TABLE 4-4 lists the `ActionRes` public data members.

TABLE 4-4 ActionRes Public Data Members

Type	Variables	Description
AsnlValue	curr_time	An optional parameter specifying the time that this response message was generated.
AsnlValue	action_type	The type of action for which this response is being generated.
AsnlValue	action_reply	Information accompanying this action response. The contents of this optional parameter vary and are based on the action type specified. The formats for this parameter for the various action types are given in the GDMO and ASN.1 documents.

4.6.1 Constructor

```
ActionRes()
```

The constructor for `ActionRes` takes no parameters. It only initializes its parent classes.

4.7 AssocReleased Class

Inheritance: public ResMess, public Message, public QueueElem

#include <pmi/message.hh>

Method Types: No public member functions are declared in this class.

The AssocReleased class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message initiator-releasing. The usage of this message is described in detail in the documentation covering the ROSE protocol. It is used within the Solstice EM MIS to indicate that a request could not be serviced because the association on which that request was received is either about to or has already gone away.

4.7.1 Constructor

`AssocReleased()`

This constructor takes no parameters. It initializes its parent classes.

4.8 CancelGetReq Class

```
Inheritance: public ReqMess, public Message, public QueueElem
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the CancelGetReq class adds a `MessId` parameter. This parameter specifies the ID of the CMIS Get request that is being cancelled by this request.

TABLE 4-5 lists the CancelGetReq public variable.

TABLE 4-5 CancelGetReq Public Variable

Type	Variable	Description
MessId	get_id	The ID of the CMIS Get request that is being cancelled by this request

4.8.1 Constructor

```
CancelGetReq()
```

This constructor takes no parameters. It only initializes its parent classes.

4.9 CancelGetRes Class

Inheritance: public ResMess, public Message, public QueueElem

#include <pmi/message.hh>

Method Types: No public member functions are declared in this class.

The CancelGetRes object class contains all of the member variables and member functions that are present in the classes it has derived from, whether directly or indirectly. No additional parameters are available for this response message.

4.9.1 Constructor

```
CancelGetRes()
```

This constructor takes no parameters. It only initializes its parent classes.

4.10 ClassInstConfl Class

```
Inheritance:public ResMess, public Message, public QueueElem
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the ClassInstConfl class adds two Asn1Value parameters to store an object class and an object instance.

TABLE 4-6 lists the ClassInstConfl public data members.

TABLE 4-6 ClassInstConfl Public Data Members

Type	Variables	Description
Asn1Value	oc	The object class that was specified in the request message
Asn1Value	oi	The object instance, whose object class is not the same as oc, that caused the generation of this error message

4.10.1 Constructor

ClassInstConfl()

This constructor takes no parameters. It only initializes its parent classes.

4.11 CreateReq Class

Inheritance: public ObjReqMess, public ReqMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the `CreateReq` class adds four `Asn1Value` parameters to store a superior object instance, access control information, a reference object instance, and an attribute list.

TABLE 4-7 lists the `CreateReq` public data members.

TABLE 4-7 CreateReq Public Data Members

Type	Variables	Description
Asn1Value	superior_oi	The object that will be the parent object (in the MIT) to this newly created object.
Asn1Value	access	Access control information that is checked by the destination to determine if the issuer of this request message is allowed to perform a creation. This parameter is optional.
Asn1Value	reference_oi	The optional object instance of an object whose attributes are to be copied into this new object.
Asn1Value	attr_list	An optional list of attributes that the newly created object is to contain.

4.11.1 Constructor

```
CreateReq()
```

This constructor takes no parameters. It initializes its parent classes and its internal data.

4.12 CreateRes Class

Inheritance: public ObjResMess, public Message, public QueueElem
 #include <pmi/message.hh>

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the CreateRes class adds two AsnlValue parameters to store a current time and an attribute list.

TABLE 4-8 lists the CreateRes public data members.

TABLE 4-8 CreateRes Public Data Members

Type	Variables	Description
AsnlValue	curr_time	An optional parameter specifying the time that this response message was generated.
AsnlValue	attr_list	This optional parameter contains a list of attribute IDs and values with which the new object was created.

4.12.1 Constructor

```
CreateRes()
```

This constructor takes no parameters. It initializes its parent classes.

4.13 DeleteReq Class

Inheritance: public ScopedReqMess, public ObjReqMess, public ReqMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

The DeleteReq object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. It does not add any parameters other than those that it inherits.

4.13.1 Constructor

```
DeleteReq()
```

This constructor takes no parameters. It only initializes its parent classes.

4.14 DeleteRes Class

Inheritance: public ObjResMess, public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the DeleteRes class adds an AsnlValue parameter to store a current time.

TABLE 4-9 lists the DeleteRes public data member.

TABLE 4-9 DeleteRes Public Data Member

Type	Variable	Description
AsnlValue	curr_time	An optional parameter specifying the time that this response message was generated

4.14.1 Constructor

```
DeleteRes()
```

This constructor takes no parameters. It only initializes its parent classes.

4.15 DuplicateOI Class

```
Inheritance: public ResMess, public Message, public QueueElem
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the DuplicateOI class adds an object instance parameter.

TABLE 4-10 lists the DuplicateOI public variable.

TABLE 4-10 DuplicateOI Public Variable

Type	Variable	Description
AsnlValue	oi	This is an invalid object instance that caused the generation of this error message. This is sent in response to a create request when the object whose creation was requested was given the same object instance as an already existing object.

4.15.1 Constructor

DuplicateOI()

This constructor takes no parameters. It only initializes its parent classes.

4.16 DupMessageId Class

Inheritance: public ResMess, class Message, class QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

The DupMessageId object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. This class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message duplicate-invocation.

4.16.1 Constructor

```
DupMessageId( )
```

This constructor takes no parameters. It only initializes its parent classes.

4.17 ErrorResUnexp Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

The `ErrorResUnexp` object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. This class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message error-response-unexpected. It is used to indicate that an error message was generated in response to a non-confirmed request.

4.17.1 Constructor

```
ErrorResUnexp( )
```

This constructor takes no parameters. It only initializes its parent classes.

4.18 EventReq Class

Inheritance: public ObjReqMess, public ReqMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the EventReq class adds three AsnlValue parameters to store an event type, an event time, and some event information.

TABLE 4-11 lists the EventReq public data members.

TABLE 4-11 EventReq Public Data Members

Type	Variables	Description
AsnlValue	event_type	The type of event reported via this message.
AsnlValue	event_time	The time that the originator of the event chose to place here (probably, but not necessarily the time that the event occurred).
AsnlValue	event_info	Any supplemental information that is to accompany this request. Specific data formats for this parameter depend on the event_type and are defined in OSI Network Management Forum documentation.

4.18.1 Constructor

```
EventReq( )
```

This constructor takes no parameters. It only initializes its parent classes.

4.19 GetListErr Class

Inheritance: public ObjResMess, public Message, public QueueElem
#include <pmi/message.hh>

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the GetListErr class adds two Asn1Value parameters to store a current time and a get an information list.

TABLE 4-12 lists the GetListErr public data members.

TABLE 4-12 GetListErr Public Data Members

Type	Variables	Description
Asn1Value	get_info_list	The list of attributes that were requested in the Get request. This list contains attributes which could be accessed. Those attributes which could not be accessed, and hence caused the generation of this error message, are not included in this list.
Asn1Value	curr_time	An optional parameter specifying the time that this response message was generated.

4.19.1 Constructor

GetListErr()

This constructor takes no parameters. It only initializes its parent classes.

4.20 GetReq Class

Inheritance: public ScopedReqMess, public ObjReqMess, public ReqMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the GetReq class adds an attribute ID list parameter.

TABLE 4-13 lists the GetReq public variable.

TABLE 4-13 GetReq Public Variable

Type	Variable	Description
AsnlValue	attr_id_list	A list of attribute IDs whose attribute values are to be returned in the response to this Get request.

4.20.1 Constructor

```
GetReq( )
```

This constructor takes no parameters. It only initializes its parent classes.

4.21 GetRes Class

Inheritance: public ObjResMess, public ResMess, public Message, public QueueElem

#include <pmi/message.hh>

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the GetRes class adds two ASN1Value parameters to store a current time and an attribute list.

TABLE 4-14 lists the GetRes public data members.

TABLE 4-14 GetRes Public Data Members

Type	Variables	Description
Asn1Value	curr_time	An optional parameter specifying the time that this response message was generated.
Asn1Value	attr_list	A list of attributes was specified in the Get request message for which this response is being generated. This parameter represents the list of attributes that were compiled and are being returned to the requestor.

4.21.1 Constructor

GetRes()

This constructor takes no parameters. It only initializes its parent classes.

4.22 InvalidActionArg Class

```
Inheritance: public ResMess, public Message, public QueueElem  
  
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the InvalidActionArg class adds two Asn1Value parameters to store a current time and some action information. There are two choices defined for the invalidArgument error message and this class defines the actionValue choice (first of the two). The oi and curr_time members are only defined when returning an error from a scoped ACTION_REQ.

TABLE 4-15 lists the InvalidActionArg public data members.

TABLE 4-15 InvalidActionArg Public Data Members

Type	Variables	Description
Asn1Value	action_info	Additional information about the action, revealing why this error message was generated.
Asn1Value	curr_time	An optional parameter specifying the time that this response message was generated.

4.22.1 Constructor

InvalidActionArg()

This constructor takes no parameters. It only initializes its parent classes.

4.23 InvalidAttrVal Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the InvalidAttrVal class adds an attribute parameter.

TABLE 4-16 lists the InvalidAttrVal public variable.

TABLE 4-16 InvalidAttrVal Public Variable

Type	Variable	Description
Asn1Value	attr	This is the invalid attribute that caused the generation of this error message.

4.23.1 Constructor

```
InvalidAttrVal()
```

This constructor takes no parameters. It only initializes its parent classes.

4.24 InvalidEventArgs Class

```
Inheritance: public ResMess, public Message, public QueueElem
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the `InvalidEventArgs` class adds three `Asn1Value` parameters to store an object class, event type, and some event information. There are two choices defined for the `invalidArgument` error message and this class defines the `eventValue` choice (second of the two).

TABLE 4-17 lists the `InvalidEventArgs` public data members.

TABLE 4-17 InvalidEventArgs Public Data Members

Type	Function	Description
Asn1Value	oc	The object that the event report request was generated for.
Asn1Value	event_type	The invalid event type that caused the generation of this error message.
Asn1Value	event_info	Additional information indicating why this error message was generated. The format of this parameter is variable and depends on the <i>event_type</i> specified in this message.

4.24.1 Constructor

InvalidEventArgs()

This constructor takes no parameters. It only initializes its parent classes.

4.25 InvalidFilter Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the InvalidFilter class adds a filter parameter.

TABLE 4-18 lists the InvalidFilter public variable.

TABLE 4-18 InvalidFilter Public Variable

Type	Variable	Description
Asn1Value	filter	The filter that was invalid and caused the generation of this error message.

4.25.1 Constructor

```
InvalidFilter()
```

This constructor takes no parameters. It only initializes its parent classes.

4.26 InvalidOI Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the InvalidOI class adds an object instance parameter.

TABLE 4-19 lists the InvalidOI public variable.

TABLE 4-19 InvalidOI Public Variable

Type	Variable	Description
AsnlValue	oi	The invalid object instance that caused the generation of this error message.

4.26.1 Constructor

```
InvalidOI()
```

This constructor takes no parameters. It only initializes its parent classes.

4.27 InvalidOperation Class

Inheritance: public ResMess, public Message, public QueueElem

#include <pmi/message.hh>

Method Types: No public member functions are declared in this class.

The InvalidOperation object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. This class defines no functions or variables beyond those it inherits.

4.27.1 Constructor

`InvalidOperation()`

This constructor takes no parameters. It only initializes its parent classes.

4.28 InvalidOperator Class

```
Inheritance:public ResMess, public Message, public QueueElem
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the InvalidOperator class adds a modify operator parameter.

TABLE 4-20 lists the InvalidOperator public variable.

TABLE 4-20 InvalidOperator Public Variable

Type	Variable	Description
AsnlValue	mod_op	The invalid modify operator that was specified in a Set request and thus caused the generation of this error message.

4.28.1 Constructor

InvalidOperator()

This constructor takes no parameters. It only initializes its parent classes.

4.29 InvalidScope Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

In addition to the functions or variables it inherits, the InvalidScope class adds a scope parameter.

TABLE 4-21 lists the InvalidScope public variable.

TABLE 4-21 InvalidScope Public Variable

Type	Variable	Description
MessScope	scope	The invalid scope parameter, extracted from the request message, which caused the generation of this error message.

4.29.1 Constructor

```
InvalidScope()
```

This constructor takes no parameters. It only initializes its parent classes.

4.30 LinkedResUnexp Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

The `LinkedResUnexp` object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. This class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message “linked-response-unexpected.” It is used within the Solstice EM MIS to indicate that a linked request was received and the linked ID specified did not refer to a request for which a linked ID could be generated.

4.30.1 Constructor

```
LinkedResUnexp()
```

This constructor takes no parameters. It only initializes its parent classes.

4.31 Message Class

Inheritance: public QueueElem

```
#include <pmi/message.hh>
```

The Message class is the base class used by almost all messages passed between SAPs and the PMI. The messages contained in the message.hh file largely define the syntax of the low-level usage of the PMI.

TABLE 4-22 lists the Message class public data members.

TABLE 4-22 Message Class Public Data Members

Type	Variables	Description
MessId	id	The Message ID, unique to the originator of the message.
Address	source	The address of the Solstice EM module or application that originated this message.
Address	dest	The address of the Solstice EM module or application that is the intended destination of this message. This field is optional when the message is first issued. If this is not supplied, the MRM determines the destination for this message and fills in this field.
MessQOS	qos	The quality of service to be used in processing this request.

TABLE 4-23 lists the Message class method types.

TABLE 4-23 Message Class Method Types

Functions	Description
type base type	Return the message's type or base type
is_request is_response is_error	Test type of message
dup	Make a copy of message
new_message	Create new message types
delete_message	Delete a message

4.31.1 Constructor

```
Message(MessType type)
```

This constructor, above, takes a `MessType` as its argument and records it as the value of the private variable *type*. For the possible values of `MessType`, refer to Section 4.69.7 “`MessType`” on page 4-84.”

4.31.2 Message Member Functions

This section describes the member functions of the `Message` class.

`basetype`

```
MessType basetype() const
```

This function call returns the message’s base type.

`delete_message`

```
static void delete_message(MessagePtr mp)
```

This function call deletes the `Message` instance to which *mp* points. Use this function as a destructor for messages.

`dup`

```
virtual MessagePtr dup() = 0
```

This function call creates a duplicate of the message and returns a pointer to the duplicate. Note that the embedded `Asn1Values` are copied by reference.

Message Class

is_error

```
Boolean is_error() const;
```

This function call returns **TRUE** if this is an error message and **FALSE** if it is not. The message is determined to be an error based on its *t* member variable.

is_request

```
Boolean is_request() const;
```

This function call returns **TRUE** if this is a request message and **FALSE** if it is not. The message is determined to be a request based on its *t* member variable.

is_response

```
Boolean is_response() const;
```

This function call returns **TRUE** if this is a response message and **FALSE** if it is not. The message is determined to be a response based on its *t* member variable.

new_message

```
static Message* new_message(MessType type)
```

This function call creates a new Message type.

type

```
MessType type() const
```

This function call returns the type of this message. For the possible values of the returned enumeration, refer to Section 4.69.7 “MessType” on page 4-84.”

4.32 MessageSAP Class

Inheritance: class MessageSAP

```
#include <pmi/message.hh>
```

A key class for the low-level use of the PMI is the MessageSAP class. The KernelMessageSAP, is a subclass of the MessageSAP class. The classes listed in TABLE 4-24 are subclasses of MessageSAP.

TABLE 4-24 MessageSAP Subclasses

Subclass Names	Description
ApplMessageSAP	A transport independent SAP used by applications
TDApplMessageSAP	A transport dependent SAP used by applications
TIMessageSAP	A transport independent SAP used by the MIS
TDMessageSAP	A transport dependent SAP used by the MIS

The MessageSAP object class is used as the endpoint of a communications link between two Solstice EM modules. A MessageSAP is created on each end of a communications path by a routine called to register a module or application with the MRM. This routine creates both MessageSaps and then notifies both parties of the MessageSAP that they are to use for communication to the other end.

The MessageSAP maintains a message ID so that request messages issued from this MessageSAP can be uniquely identified. In addition, it contains two instances of the Event object class. These two instances are the `receive_request` event and the `detach` event. These are used to notify the owner of the MessageSAP that a request message has been received and also to notify the owner of the MessageSAP that the other side (the MessageSAP to which this one is attached) has been deleted.

The MessageSAP object class defines a number of member functions which are used to either send or receive messages. In addition, member functions are provided that return a unique message ID and cancel a callback for a message or an event.

The MessageSAP class defines queues of pointers to messages. The messages pointed to are all subclasses of the Message class defined in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

FIGURE 4-3 shows the inheritance hierarchy for the classes based on MessageSAP.

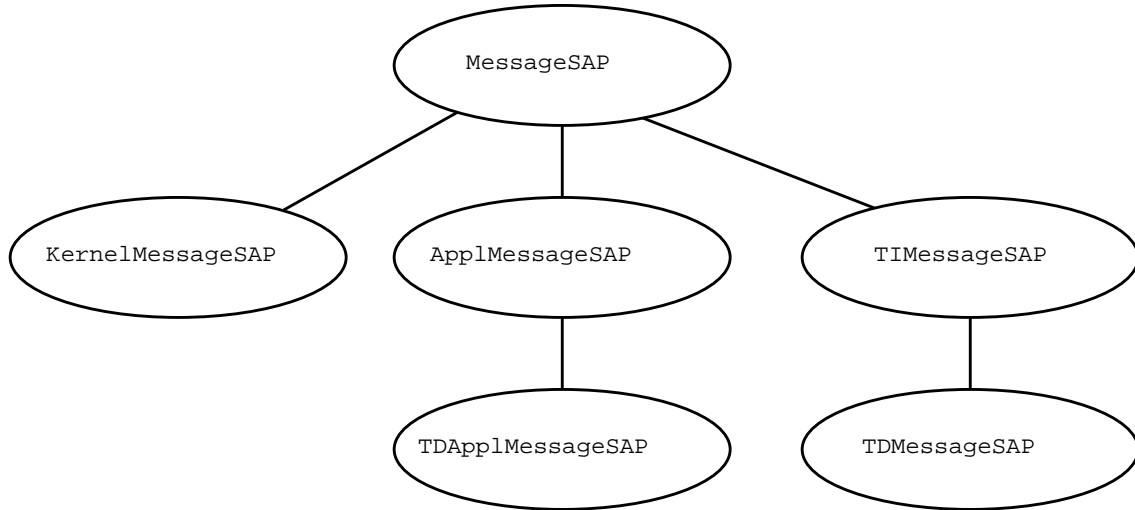


FIGURE 4-3 Inheritance Tree of the MessageSAP Class

TABLE 4-25 lists the MessageSAP public data members.

TABLE 4-25 MessageSAP Public Data Members

Type	Variables	Description
Callback	receive_request_cb	The event to be posted when an incoming request arrives.
Callback	detach_cb	The event to be posted when the message sap detaches.

TABLE 4-26 lists the MessageSAP method types.

TABLE 4-26 MessageSAP Method Types

Method Name	Method Type
send	Send a message (with or without blocking).
receive_request receive_response	Respond to a received request.
cancel_callback	Cancel the callback for a pending response.
new_id	Generate an ID for a message.

4.32.1 Constructor

```
MessageSAP ( )
```

This example is the constructor for the MessageSAP.

4.32.2 MessageSAP Member Functions

This section describes the member functions of the MessageSAP class.

cancel_callback

```
virtual void cancel_callback(MessId m_id) = 0;
```

This function call cancels the pending response callback for a particular message ID. This form of the `cancel_callback` member function cancels the callback that was attached to a request sent via the non-blocking form of the `send` member function. This function takes a message id *m_id* as input and searches for any callback routine which might have been specified for the response identified by *m_id*. The callback is removed from the list of callbacks so that when the response message arrives, it is dropped. It returns OK if the callback is successfully cancelled.

Alternatively,

```
virtual void cancel_callback(Callback &cb) = 0;
```

This function call cancels the pending response callback for any matching message. This form of the `cancel_callback` member function cancels the callback that was attached to a request sent via the non-blocking form of the `send` member function. This function takes an event reference *e* as input and searches references to this event in the callback list. If any are found, they are deleted from the list. Later, when the response message arrives, it is dropped.

`new_id`

```
MessId new_id()
```

This function call generates a new message identifier (one greater than the last ID this MessageSAP supplied), stores it privately in the MessId instance, and returns its value.

`receive_request`

```
virtual Result receive_request(MessagePtr &mp) = 0;
```

This function call receives the next pending request message. This function is called after a notification has arrived via the `receive_request_callback` mechanism. The `receive_request_callback` receives a notification that a request message has been queued up for this MessageSAP and then this routine should be called to actually access the message. The function takes a reference to a Message pointer and this pointer is set to point to the message received.

```
virtual Result receive_response(MessId m_id,  
    MessagePtr &mp) = 0;  
virtual Result receive_response(ResponseHandle rh,  
    MessagePtr &mp) = 0;
```

This function call receives the next response for a given message. The message whose response is sought is identified either by its message ID (*m_id*) or by a response handle (*rh*). This function is called after the owner of the MessageSAP has been notified that a response message has been queued to this MessageSAP. The notification takes place by having the blocking form of the MessageSAP::send return successful for a confirmed request message. This notification can also occur if the event for the non-blocking form of the send member function is notified. The function sets *mp* with a pointer to the response message. It returns OK if the first argument successfully identifies a response and the message pointer is successfully set to the response message.

send

```
virtual SendResult send( MessagePtr mp,
    MTime block_time = INFINITY) = 0;
```

This function call is the blocking form of the `send` member function. This version of `send` takes a pointer to an instance of the `Message` object class and a *block_time* parameter. The message pointer points to the message that is to be sent via this `MessageSAP`. The *block_time* parameter specifies how long the caller of this function is willing to wait for the message to be sent. If the message cannot be sent within the time specified, an error is returned.

Messages might not be sent because of resource limitations or a host of other problems.

The above function call returns an error code that specifies why a message could not be sent. The possible values of `SendResult` are shown below.

```
typedef enum SendResult { SENT,
    BAD_MESSAGE,
    WOULD_BLOCK,
    NO_MEM };
```

If this is a request message and the function returns `SENT`, the request message has been successfully sent and the response to this request has been queued to this `MessageSAP`. If this was a non-confirmed request, `SENT` indicates only that the request has been sent successfully.

If this is a response message, `SENT` indicates that the response has been sent.

The following function call is the non-blocking form of the `send` member function. This version of `send` takes a pointer to an instance of the `Message` object class, an instance of the `Event` object class, and a *block_time* parameter. This version of `send` should only be used to send confirmed request messages. It should not be used to send unconfirmed requests or responses.

The message pointer *mp* points to the message that is to be sent via this MessageSAP. The callback *cb* specifies a procedure that is to be called whenever the response for this request message has been queued to this MessageSAP. The *block_time* parameter specifies how long the caller of this function is willing to wait for the message to be sent. If the message cannot be sent within the time specified, an error is returned.

```
virtual SendResult send( MessagePtr mp,
                        const Callback &cb,
                        MTime block_time = INFINITY) = 0;
```

It might not be possible to send a message because of resource limitations or a host of other problems. This routine returns an error code that indicates why a message could not be sent.

4.32.3 MessageSAP Initialization

Following is a sample on how to initialize a MessageSAP.

```
result init_kernel_msg_sap(Address source_module,
                          MessageSAP **sap_p_p);
```

The parameter, *sap_p_p*, is initialized as a result of the call to `init_kernel_msg_sap`. It is the MessageSAP your driver module should use to send and receive messages from the Solstice EM MRM. After `init_kernel_msg_sap` has been called, the MessageSAP parameter should be initialized to point to the callback handlers that are used by the MRM.

The function returns a value whose type is `Result`: that is, a boolean value defined as either `OK` or `NOT_OK`. `Result` is defined in the `/opt/SUNWconn/em/include/pmi/sys_type.hh` file.

4.33 MessQOS Class

Inheritance: `class MessQOS`

`#include <pmi/message.hh>`

Method Types: No public member functions are declared in this class.

The `MessQOS` class represents the Quality of Service indicator included in all messages. This class is currently a null class (that is, it has no member functions or variables). Its purpose is to store data that affects the type of service to be given to a message. This data might include such things as the allowable lifetime of the message and the type of behavior associated with the transportation of this message. Some examples are:

- request time-out parameters
- security parameters
- reliability/retry parameters

4.34 MessScope Class

Inheritance: `class MessScope`

`#include <pmi/message.hh>`

Method Types: No method types are declared in this class.

The `MessScope` class defines a message's scope (that is, the range of objects to which the message is to be applied). An instance of `MessScope` contains a `MessScopeType` variable and an optional level. The enumeration defines five types of scoping.

TABLE 4-27 Types of `MessScope` Scoping

Scope	Description
<code>BASE_OBJECT</code>	A request message containing a scope parameter equal to <code>BASE_OBJECT</code> should be sent only to the object specified in the request.

TABLE 4-27 Types of MessScope Scoping (*Continued*)

Scope	Description
NTH_LEVEL	The request message should be sent to those objects which exist N levels below the base object in the Management Information Tree (MIT). The base object is not part of the message, only those objects which are Nth level descendents of the base object. The <i>level</i> variable in the MessScope class is set to the level desired.
BASE_TO_NTH_LEVEL	The request message is sent to the base object and all descendents of the base object down to the Nth level. Again, the <i>level</i> variable is used to indicate the final level of objects that the request is to be routed to.
ALL_LEVELS	The request message to be sent to the base object and all of its descendents in the MIT. The <i>level</i> variable in the MessScope class is only used for NTH_LEVEL and BASE_TO_NTH_LEVEL scoping.
ALL_LEVELS_EXCEPT_BASE	Used internally by the Solstice EM MIS.

TABLE 4-28 lists the MessScope public data members.

TABLE 4-28 MessScope Public Data Members

Type	Variables	Description
MessScopeType	type	{BASE_OBJECT, NTH_LEVEL, BASE_TO_NTH_LEVEL, ALL_LEVELS}
U32	level	

4.34.1 Constructors

This section specifies the constructors of the MessScope class.

```
MessScope( )  
MessScope(MessScopeType type, U32 level)
```

4.35 MissingAttrVal Class

Inheritance: public ResMess, public Message, public QueueElem

#include <pmi/message.hh>

Method Types: The public methods Message::new_message, Message::delete_message, and MessagePtr dup() are defined in this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the MissingAttrVal class adds an attribute ID list parameter.

TABLE 4-29 lists the MissingAttrVal public variable

TABLE 4-29 MissingAttrVal Public Variable

Type	Variable	Description
AsnlValue	attr_id_list	A list of attribute IDs is specified in the Create request. If an attribute is required to be in an object when the object is created and that attribute is not present in this list, then this error message is generated. This variable contains the list of attributes whose values are missing.

4.35.1 Constructor

MissingAttrVal()

This constructor takes no parameters. It only initializes its parent classes.

4.36 MistypedArg Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined in this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `MistypedArg` object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. This class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message `mistyped-argument`. The usage of this message is described in detail in the documentation covering the ROSE protocol. It is used within the Solstice EM MIS to indicate that one of the arguments supplied with a request message was not supposed to be present.

4.36.1 Constructor

```
MistypedArg()
```

This constructor takes no parameters. It only initializes its parent classes.

4.37 MistypedError Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined in this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `MistypedError` object class contains all of the member data members and member functions that are present in the classes that it has derived from, either directly or indirectly. This class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message `mistyped-parameter`. It is used within the Solstice EM MIS to indicate that an error message, generated in response to a particular request, contained a parameter which either was not expected as part of the error message or which was not formed properly.

4.37.1 Constructor

```
MistypedError()
```

This constructor takes no parameters. It only initializes its parent classes.

4.38 MistypedOp Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `MistypedOp` object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. This class does not add any additional parameters.

4.38.1 Constructor

```
MistypedOp()
```

This constructor takes no parameters. It only initializes its parent classes.

4.39 MistypedRes Class

Inheritance: public ResMess, public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `MistypedRes` object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. This class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message `mistyped-result`. It is used within the Solstice EM MIS to indicate that a result message, generated for a particular request, contained a parameter that was either not expected or was not formed properly.

4.39.1 Constructor

```
MistypedRes()
```

This constructor takes no parameters. It only initializes its parent classes.

4.40 NoSuchAction Class

Inheritance: public ObjResMess, public Message, public QueueElem
`#include <pmi/message.hh>`

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `NoSuchAction` class adds two `Asn1Value` parameters to store a current time and an action type.

The *action_type* and *curr_time* members are only defined when returning an error from a scoped `ACTION_REQ`.

TABLE 4-30 lists the `NoSuchAction` public data members.

TABLE 4-30 NoSuchAction Public Data Members

Type	Variables	Description
Asn1Value	action_type	The invalid action type as extracted from the request message.
Asn1Value	curr_time	An optional parameter specifying the time that this response message was generated.

4.40.1 Constructor

NoSuchAction()

This constructor takes no parameters. It only initializes its parent classes.

4.41 NoSuchActionArg Class

```
Inheritance:public ObjResMess, public ResMess, public Message,  
public QueueElem  
  
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `NoSuchActionArg` class adds two `Asn1Value` parameters to store a current time and an action type. There are two choices defined for the `noSuchArgument` error message and this class defines the `actionId` choice (first of the two).

The *action_type* and *curr_time* members are only defined when returning an error from a scoped `ACTION_REQ`.

TABLE 4-31 lists the `NoSuchActionArg` public data members.

TABLE 4-31 `NoSuchActionArg` Public Data Members

Type	Variables	Description
Asn1Value	action_type	The invalid action type as extracted from the request message.
Asn1Value	curr_time	An optional parameter specifying the time that this response message was generated.

4.41.1 Constructor

NoSuchActionArg()

This constructor takes no parameters. It only initializes its parent classes.

4.42 NoSuchAttr Class

Inheritance: public ResMess, public Message, public QueueElem

#include <pmi/message.hh>

Method Types: The public methods Message::new_message, Message::delete_message, and MessagePtr dup() are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the NoSuchAttr class adds an attribute ID parameter.

TABLE 4-32 lists the NoSuchAttr public variable.

TABLE 4-32 NoSuchAttr Public Variable

Type	Variable	Description
AsnlValue	attr_id	This is the invalid attribute ID that caused the generation of this error message.

4.42.1 Constructor

NoSuchAttr()

This constructor takes no parameters. It only initializes its parent classes.

4.43 NoSuchEvent Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `NoSuchEvent` class adds two `Asn1Value` parameters to store an object class and an event type.

TABLE 4-33 lists the `NoSuchEvent` public data members.

TABLE 4-33 NoSuchEvent Public Data Members

Type	Variables	Description
Asn1Value	oc	The object class for which the event report request was generated.
Asn1Value	event_type	The invalid event type that caused the generation of this error message.

4.43.1 Constructor

```
NoSuchEvent ( )
```

This constructor takes no parameters. It only initializes its parent classes.

4.44 NoSuchEventArgs Class

```
Inheritance:public ResMess, public Message, public QueueElem  
  
#include <pmi/message.hh>
```

Method Types: The public methods Message::new_message, Message::delete_message, and MessagePtr dup() are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the NoSuchEventArgs class adds two Asn1Value parameters to store an object class and an event type. There are two choices defined for the noSuchArgument error message and this class defines the eventId choice (second of the two).

TABLE 4-34 lists the NoSuchEventArgs public data members.

TABLE 4-34 NoSuchEventArgs Public Data Members

Type	Variables	Description
Asn1Value	oc	The object class for which the event report request was generated.
Asn1Value	event_type	The invalid event type that caused the generation of this error message.

4.44.1 Constructor

NoSuchEventArgs()

This constructor takes no parameters. It only initializes its parent classes.

4.45 NoSuchMessageId Class

```
Inheritance: public ResMess, public Message, public QueueElem  
  
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `NoSuchMessageId` class adds a `get ID` parameter.

TABLE 4-35 lists the `NoSuchMessageId` public variable.

TABLE 4-35 `NoSuchMessageId` Public Variable

Type	Variable	Description
AsnlValue	<i>get_id</i>	This is the invalid message ID that caused generation of this error message.

4.45.1 Constructor

NoSuchMessageId()

This constructor takes no parameters. It only initializes its parent classes.

4.46 NoSuchOC Class

Inheritance: public ResMess, public Message, public QueueElem
`#include <pmi/message.hh>`

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `NoSuchOC` class adds an `Asn1Value` parameter to store an object class.

TABLE 4-36 lists the `NoSuchOC` public variable.

TABLE 4-36 NoSuchOC Public Variable

Type	Variable	Description
Asn1Value	oc	The invalid object class that caused the generation of this error message.

4.46.1 Constructor

NoSuchOC ()

This constructor takes no parameters. It only initializes its parent classes.

4.47 NoSuchOI Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `NoSuchOI` class adds an `AsnlValue` parameter to store an object instance.

TABLE 4-37 lists the `NoSuchOI` public variable.

TABLE 4-37 NoSuchOI Public Variable

Type	Variable	Description
AsnlValue	oi	The invalid object instance that caused the generation of this error message.

4.47.1 Constructor

```
NoSuchOI ( )
```

This constructor takes no parameters. It only initializes its parent classes.

4.48 NoSuchRefOI Class

```
Inheritance:public ResMess, public Message, public QueueElem  
  
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `NoSuchRefOI` class adds an object instance parameter.

TABLE 4-38 lists the `NoSuchRefOI` public variable.

TABLE 4-38 NoSuchRefOI Public Variable

Type	Variable	Description
Asn1Value	oi	The invalid object instance that caused the generation of this error message.

4.48.1 Constructor

NoSuchRefOI()

This constructor takes no parameters. It only initializes its parent classes.

4.49 ObjReqMess Class

Inheritance: public ReqMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `ObjReqMess` class adds two `Asn1Value` parameters for an object class and an object instance.

TABLE 4-39 lists the `ObjReqMess` public data members.

TABLE 4-39 ObjReqMess Public Data Members

Type	Variables	Description
Asn1Value	oc	This is either a base or managed object class as defined for the type of message being created.
Asn1Value	oi	This is either a base or managed object instance as defined for the type of message being created.

4.49.1 Constructor

```
ObjReqMess(MessType type)
```

4.50 ObjResMess Class

Inheritance:public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `ObjResMess` class adds two `Asn1Value`s to store an object class and an object instance. Any response (also linked request, error, and linked error) messages which include object class and object instance parameters in the message would be derived from this class.

TABLE 4-40 lists the `ObjResMess` public data members.

TABLE 4-40 ObjResMess Public Data Members

Type	Variables	Description
Asn1Value	oc	The object class for this response message. .
Asn1Value	oi	The object instance for this response message.

4.50.1 Constructor

ObjResMess (MessType *type*)

This constructor takes a `MessType` variable as input and passes this variable on to the constructor(s) for the classes from which this class is derived.

4.51 OpCancelled Class

Inheritance: public ResMess, public Message, public QueueElem

#include <pmi/message.hh>

Data Members: No public data members are declared in this class.

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `OpCancelled` object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. This class defines no functions or variables beyond those it inherits.

4.51.1 Constructor

```
OpCancelled()
```

This constructor takes no parameters. It only initializes its parent classes.

4.52 ProcessFailure Class

Inheritance:public ObjResMess, public ResMess, public Message,
public QueueElem

#include <pmi/message.hh>

Method Types: The public methods Message::new_message,
Message::delete_message, and MessagePtr dup() are defined for this class.
These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the ProcessFailure class adds
a specific error information parameter.

TABLE 4-41 lists the ProcessFailure public variable.

TABLE 4-41 ProcessFailure Public Variable

Type	Function	Description
AsnlValue	spec_err_info	Error information which gives additional information about why this error message was generated. The format of this parameter is variable and depends upon the object class specified in this error message.

4.52.1 Constructor

ProcessFailure()

This constructor takes no parameters. It only initializes its parent classes.

4.53 ReqMess Class

Inheritance: public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `ReqMess` class adds a `MessMode` parameter to a message. The usage of `MessMode` variables are described in Section 4.69.2 “MessMode” on page 4-82.”

TABLE 4-42 lists the `ReqMess` public data members.

TABLE 4-42 ReqMess Public Data Members

Type	Variables	Description
MessMode	mode	The mode in which a request message is sent: {CONFIRMED, UNCONFIRMED}
Oid	app_context	The application context name for this request message. This is used to establish an association within the protocol driver.
U32	flags	enum
	ReqFlags	{OVERRIDE_NAME_BINDING = 1, OVERRIDE_ATTR_CHECKS = 2, INTERNAL_RELATIONSHIP_CHANGE = 4};

4.53.1 Constructor

```
ReqMess(MessType type)
```

This constructor takes a `MessType` variable as input and sets the member variable *mode* equal to the variable passed in.

4.54 ResMess Class

Inheritance: public Message, public QueueElem

#include <pmi/message.hh>

Method Types: The public methods Message::new_message, Message::delete_message, and MessagePtr dup() are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the ResMess class adds a Boolean variable called linked that indicates whether this is a linked response message. In all cases, a linked response contains the same data as would be found in an unlinked response so that (applied to the same message) linked_get_result returns the same data as get_result. Error messages are considered to be response messages, but can also be linked requests.

TABLE 4-43 lists the ResMess public variable.

TABLE 4-43 ResMess Public Variable

Type	Variable	Description
Boolean	linked	When this variable is set to TRUE, this is a linked request message. When FALSE, this is a response message.

4.54.1 Constructor

ResMess(MessType type)

This constructor takes a MessType variable as input and passes this variable on to the constructor(s) for the classes from which this class is derived.

4.55 ResourceLimit Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `ResourceLimit` object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. This class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message resource-limitation. It is used within the Solstice EM MIS to indicate that the receiver of a request message was unable to service the request due to a lack of resources.

4.55.1 Constructor

```
protected ResourceLimit()
```

This constructor takes no parameters. It only initializes its parent classes.

4.56 ScopedReqMess Class

Inheritance: public ObjReqMess, public ReqMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `ScopedReqMess` class adds a `MessScope` variable, two `Asn1Value` parameters specifying a filter and access control, and a `MessSync` parameter. These parameters are present in all request messages for which scoping can be specified. Each of these parameters is optional.

TABLE 4-44 lists the `ScopedReqMess` public data members.

TABLE 4-44 ScopedReqMess Public Data Members

Type	Variables	Description
MessScope	scope	The type of scoping to be used for this request message. The possible scoping types are listed in Section 4.69.4 “MessScopeType” on page 4-82.”
Asn1Value	filter	This defines a filter that all objects selected via scoping must pass. The message is not sent to any object that does not pass the filter.
Asn1Value	access	This defines the access control that objects selected via scoping must be pass. The message is not sent to any object that does not pass the access control.
MessSync	sync	The type of synchronization for this scoped message; {ATOMIC, BEST_EFFORT}

4.56.1 Constructor

```
ScopedReqMess(MessType type)
```

This constructor takes a `MessType` variable as input and passes this variable on to the constructor(s) for the classes from which this class derived.

4.57 SetListErr Class

Inheritance: public ObjResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `SetListErr` class adds two `Asn1Value` parameters to store a current time and a set information list. TABLE 4-45 lists the `SetListErr` public data members.

TABLE 4-45 SetListErr Public Data Members

Type	Variables	Description
Asn1Value	set_info_list	The list of attributes slated for modification by the Set request. This list contains any attributes which could be modified as well as any attributes which were in error and thus caused the generation of this error message.
Asn1Value	curr_time	An optional parameter specifying the time that this response message was generated.

4.57.1 Constructor

```
SetListErr()
```

This constructor takes no parameters. It only initializes its parent classes.

4.58 SetReq Class

Inheritance: public ScopedReqMess, public ReqMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `SetReq` object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. In addition to the functions or variables it inherits, the `SetReq` class adds an attribute list parameter.

TABLE 4-46 lists the `SetReq` public variable.

TABLE 4-46 SetReq Public Variable

Type	Variable	Description
Asn1Value	modify_list	Each element of this list contains an attribute ID, an attribute value, and a modify operator.

4.58.1 Constructor

```
SetReq()
```

This constructor takes no parameters. It only initializes its parent classes.

4.59 SetRes Class

Inheritance: public ObjResMess, public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `SetRes` class adds two `Asn1Value` parameters to store a current time and an attribute list. TABLE 4-47 lists the `SetRes` public data members.

TABLE 4-47 SetRes Public Data Members

Type	Variables	Description
Asn1Value	curr_time	An optional parameter specifying the time that this response message was generated.
Asn1Value	attr_list	A list of attributes was specified in the Set request message for which this response is being generated. This parameter basically echoes back to the requester the list of attributes that were modified and their new values.

4.59.1 Constructor

```
SetRes()
```

This constructor takes no parameters. It only initializes its parent classes.

4.60 SyncNotSupp Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

In addition to the functions or variables it inherits, the `SyncNotSupp` class adds a `sync` parameter.

Each of the classes derived from `Message` relies on the ISO specifications of the CMIP protocol and ASN.1 data encoding. TABLE 4-48 lists the `SyncNotSupp` public variable.

TABLE 4-48 SyncNotSupp Public Variable

Type	Variable	Description
MessSync	sync	Specifies the type of synchronization which was not able to be performed and caused the generation of this error message.

4.60.1 Constructor

SyncNotSupp()

This constructor takes no parameters. It only initializes its parent classes.

4.61 TimedOut Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

This class defines no functions or variables beyond those it inherits. The error message that this class represents is generated whenever the life-span of a message has been exceeded. Future functionality envisioned for the Solstice EM MIS would be to include in the `MessQOS` (quality of service) class some indication of how long the requester is willing to wait for a response to a given request message (a message lifetime). A `TimedOut` message would be generated whenever the lifetime for a given request message had been exceeded (that is, whenever the request has not been responded to within its lifetime).

4.61.1 Constructor

```
TimedOut()
```

This constructor takes no parameters. It only initializes its parent classes.

4.62 UnexpChildOp Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `UnexpChildOp` object class contains all of the member variables and member functions that are present in the classes that it has derived from, either directly or indirectly. This class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message unexpected-child-operation. It is used within the Solstice EM MIS to indicate that a linked request was received and that the linked ID specified did not refer to a request for which this type of linked reply is valid.

4.62.1 Constructor

```
UnexpChildOp()
```

This constructor takes no parameters. It only initializes its parent classes.

4.63 UnexpError Class

Inheritance: `class UnexpError : public ResMess, public Message, public QueueElem`

`#include <pmi/message.hh>`

Data Members: No public data members are declared in this class.

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `UnexpError` class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message `unexpected-error`. It is used to indicate that an error message, generated in response to a particular request, is not one of the set of error messages that can be sent in response to that request.

4.63.1 Constructor

```
UnexpError()
```

This constructor takes no parameters. It only initializes its parent classes.

4.64 UnexpRes Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Data Members: No public data members are declared in this class.

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `UnexpRes` object class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message “result-response-unexpected”. It is used to indicate that a result message was generated for as non-confirmed request.

4.64.1 Constructor

`UnexpRes()`

This constructor takes no parameters. It only initializes its parent classes.

4.65 UnrecError Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: No public member functions are declared in this class.

The `UnrecError` class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message “unrecognized-error”. It is used to indicate that an error message, generated in response to a particular request, is not one of the set of error messages known within Solstice EM.

4.65.1 Constructor

```
UnrecError()
```

This constructor takes no parameters. It only initializes its parent classes.

4.66 UnrecLinkId Class

Inheritance: public ResMess, public Message, public QueueElem

#include <pmi/message.hh>

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `UnrecLinkId` class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message “unrecognized-linked-id.” It is used to indicate that a linked request could not be serviced because the linked ID specified in the request did not refer to any known outstanding request.

4.66.1 Constructor

`UnrecLinkId()`

This constructor takes no parameters. It only initializes its parent classes.

4.67 UnrecMessageId Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `UnrecMessageId` class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message “unrecognized-invocation”. It is used to indicate that a result or error message was generated where the message ID specified did not refer to any outstanding request.

4.67.1 Constructor

```
UnrecMessageId( )
```

This constructor takes no parameters. It only initializes its parent classes.

4.68 UnrecOp Class

Inheritance: public ResMess, public Message, public QueueElem

```
#include <pmi/message.hh>
```

Method Types: The public methods `Message::new_message`, `Message::delete_message`, and `MessagePtr dup()` are defined for this class. These classes are described in Section 4.31 “Message Class” on page 4-36.”

The `UnrecOp` object class defines no functions or variables beyond those it inherits. The error message that this class represents is patterned after the ROSE error message “unrecognized operation.” The usage of this message is described in detail in the documentation covering the ROSE protocol. It is used to indicate that the operation requested is not known to the receiver of the request.

4.68.1 Constructor

`UnrecOp()`

This constructor takes no parameters. It only initializes its parent classes.

4.69 Constants and Defined Types

The following subsections describe the constants and defined types for the low-level usage of the PMI.

- "MessId" on page 81
- "MessMode" on page 82
- "MessagePtr" on page 82
- "MessScopeType" on page 82
- "MessSync" on page 83
- "MessBaseType" on page 83
- "MessType" on page 84
- "MESSTYPE_MAX" on page 86
- "ResponseHandle" on page 86
- "SendResult" on page 86

4.69.1 MessId

A `MessId` variable is included in each message passed by way of a `MessageSAP`. The `MessId` variable is used to uniquely identify outstanding request messages for a Solstice EM module or application. Each Solstice EM module is responsible for generating new request messages that have unique message IDs. Unique means that the message can not have the same ID as another request (that is, still outstanding from this module or applicatio). The `MessId` variable is declared in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

```
typedef I32 MessId;
```

4.69.2 MessMode

The `MessMode` variable is declared in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

```
enum MessMode
{
    CONFIRMED,
    UNCONFIRMED };
```

4.69.3 MessagePtr

The `MessagePtr` variable is declared in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

```
typedef class Message *MessagePtr;
```

4.69.4 MessScopeType

The `MessScopeType` variable is declared in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

```
enum MessScopeType
{
    BASE_OBJECT,
    NTH_LEVEL,
    BASE_TO_NTH_LEVEL,
    ALL_LEVELS,
    ALL_LEVELS_EXCEPT_BASE };
```


4.69.5 MessSync

This enumeration defines the types of message synchronization that can be requested: `ATOMIC` or `BEST_EFFORT`. Currently only `BEST_EFFORT` is supported, since the requirements for `ATOMIC` synchronization are not fully defined by the standards organizations.

Type of Synchronization	Description
<code>BEST_EFFORT</code>	Scoped requests are to be processed in a best-effort fashion; if one part of a scoped request fails, the other parts of the scoped request are still attempted.
<code>ATOMIC</code>	If it were operational, would indicate that if any portion of a scoped request failed, subsequent parts of the request should not be attempted and already completed parts should be reversed.

The `MessSync` variable is declared in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

```
enum MessSync
{
    BEST_EFFORT,
    ATOMIC };
```

4.69.6 MessBaseType

The `MessBaseType` variable is declared in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

```
enum MessBaseType
{
    MESSAGE,
    REQ_MESS,
    OBJ_REQ_MESS,
    SCOPED_REQ_MESS,
    RES_MESS,
    OBJ_RES_MESS };
```

4.69.7 MesSType

The `MesSType` enumeration defines a unique ID for each type of message that can be sent within Solstice EM. This includes all the CMIS request messages, CMIS response messages, CMIS error messages, ROSE user-reject responses, SNMP requests, SNMP responses, and Solstice EM error responses. CODE EXAMPLE 4-1 provides an exhaustive list of the messages it is possible to send via the `MessageSAP` interface. Each of these messages derives from one of the base message classes defined in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

CODE EXAMPLE 4-1 MessageSAP Messages

```
enum MesSType
{
    // requests
    EVENT_REPORT_REQ
    GET_REQ
    SET_REQ
    ACTION_REQ
    CREATE_REQ
    DELETE_REQ
    CANCEL_GET_REQ

    // normal responses
    EVENT_REPORT_RES
    GET_RES
    SET_RES
    ACTION_RES
    CREATE_RES
    DELETE_RES
    CANCEL_GET_RES

    // Errors
    NO_SUCH_OC
    NO_SUCH_OI
    ACCESS_DENIED
    SYNC_NOT_SUPP
    INVALID_FILTER
    NO_SUCH_ATTR
    INVALID_ATTR_VAL
    GET_LIST_ERR
    SET_LIST_ERR
    NO_SUCH_ACTION
    PROCESS_FAILURE
    DUPLICATE_OI
    NO_SUCH_REF_OI
}
```

CODE EXAMPLE 4-1 MessageSAP Messages (*Continued*)

```

NO_SUCH_EVENT
NO_SUCH_ACTION_REQ
NO_SUCH_EVENT_ARG
INVALID_ACTION_ARG
INVALID_SCOPE
INVALID_OI
MISSING_ATTR_VAL
CLASS_INST_CONFL
COMPLEX_LIMIT
MISTYPED_OP
INVALID_OPERATION
INVALID_OPERATOR
NO_SUCH_MESSAGE_ID
OP_CANCELLED

// ROSE level user-reject responses
DUP_MESSAGE_ID
UNREC_OP
MISTYPED_ARG
RESOURCE_LIMIT
ASSOC_RELEASED
UNREC_LINKED_ID
LINKED_RES_UNEXP
UNEXP_CHILD_OP
UNREC_MESSAGE_ID
UNEXP_RES
MISTYPED_RES
ERROR_RES_UNEXP
UNREC_ERROR
UNEXP_ERROR
MISTYPED_ERROR

// Solstice EM error responses
TIMED_OUT
DEST_UNREACH
NO_SUCH_DEST
};

```

4.69.8 MESSTYPE_MAX

The `MESSTYPE_MAX` variable is declared in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

```
const MessageType MESSTYPE_MAX = NO_SUCH_DEST;
```

4.69.9 ResponseHandle

The `ResponseHandle` variable is declared in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

```
typedef void *ResponseHandle;
```

4.69.10 SendResult

The `SendResult` variable is declared in the `/opt/SUNWconn/em/include/pmi/message.hh` file.

```
typedef enum SendResult
{
    SENT,
    BAD_MESSAGE,
    WOULD_BLOCK,
    NO_MEM };

// Used by the MessageSAP class.

#define SENT TRUE
```

Access Control API

To access to Solstice EM tools and managed objects, users must belong to a group. In addition, users' access privileges are determined based on the group to which they belong. The Access Control API provides a solid C++ interface to GDMO object classes.

The Access Control API enables you to:

- Assign rules that define access for groups of users
- Define the access rules at a group level
- Control access to Solstice EM applications and managed objects

This chapter comprises the following topics:

- Section 5.1 “Design Objectives” on page 5-1
- Section 5.2 “Access Control Types” on page 5-2
- Section 5.3 “Class Hierarchy” on page 5-2
- Section 5.4 “Symbolic Constants and Defined Types” on page 5-4
- Section 5.5 “Access Control API Classes” on page 5-11

5.1 Design Objectives

The Access Control API was developed with the following design objectives:

- Uniform treatment of all the access control objects to maximize reusability of design and code
- Compatibility with the X.741 standard
- Ease of use, especially for users familiar with PMI
- Object-oriented design of the API, so that it is consistent with the other Solstice EM APIs

5.2 Access Control Types

The Access Control API defines two types of access control:

- **Object-level access control.** Controls the level of access to managed objects. For example, users belonging to the operator group are denied access to the log object, but they get back a response.
- **Feature-level access control.** Controls the level of access to application features. For example, users belonging to the operator group are denied access to the destroy feature of the `em_viewer` application, but allowed to access the view feature.

5.3 Class Hierarchy

FIGURE 5-1 illustrates the hierarchy of the Access Control API container classes.

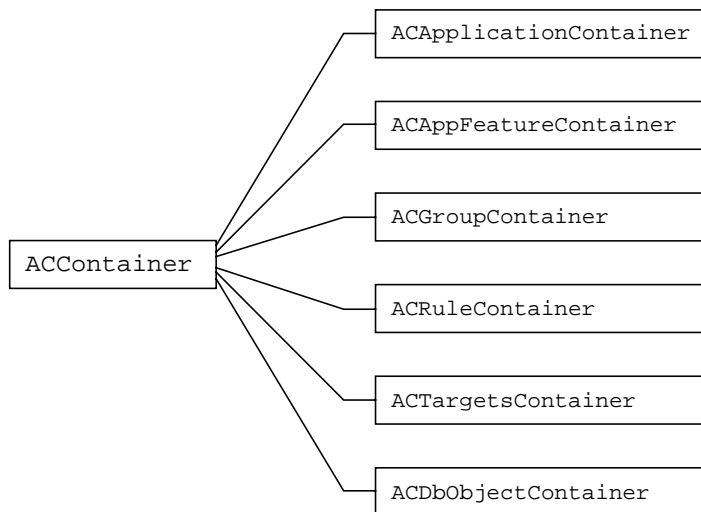


FIGURE 5-1 C++ Container Classes and Their Inheritance

FIGURE 5-2 Illustrates the hierarchy of the Access Control API.

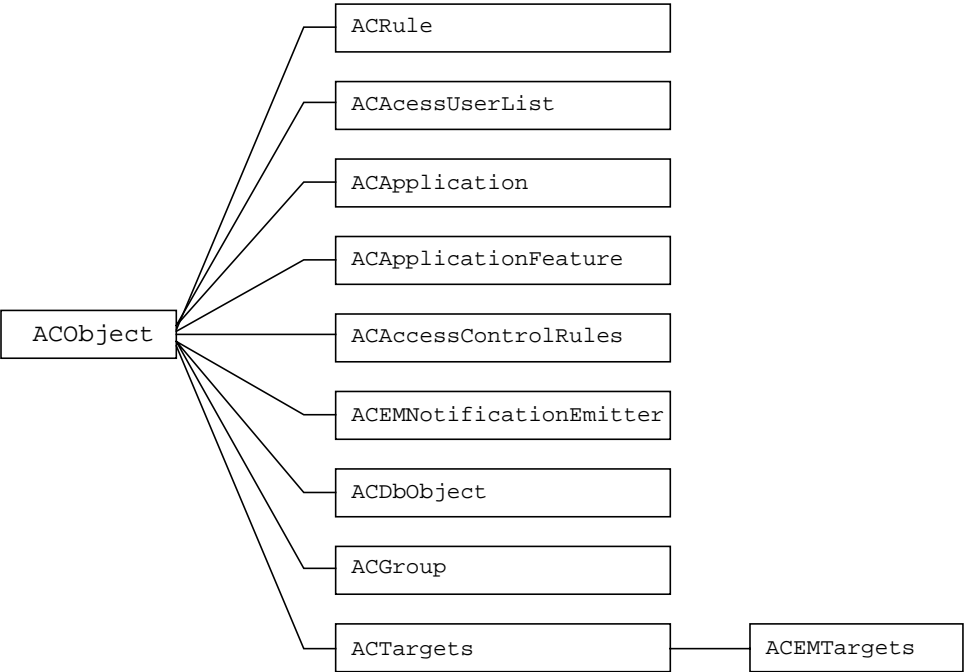


FIGURE 5-2 Access Control C++ Classes and Their Inheritance

TABLE 5-1 describes the Access Control API classes.

TABLE 5-1 Access Control API Classes

Class	Description
ACAccessControlRules Class	Represents the emAccessControlRules GDMO object class.
ACAccessUserList Class	Represents the accessUserList GDMO object class.
ACAppFeatureContainer Class	Contains all features of an application.
ACAApplication Class	Represents the application GDMO object class.
ACAApplicationContainer Class	Contains all applications.
ACAApplicationFeature Class	Represents the applicationFeature GDMO object class.
ACCallback Class	Extension of the PMI Callback class.
ACContainer Class	Abstract base class for C++ container objects.

Symbolic Constants and Defined Types

TABLE 5-1 Access Control API Classes (*Continued*)

Class	Description
ACDbObject Class	Represents the emDbObject GDMO object class.
ACDbObjectContainer Class	Represents the emDbInfo GDMO object class.
ACEMNotificationEmitter Class	Represents the emNotificationEmitter GDMO object class.
ACEMTargets Class	Represents the emTargets GDMO object class.
ACGroup Class	Represents the group GDMO object class.
ACGroupContainer Class	Represents the groupContainer object class.
ACInterface Class	Contains all container objects. Main single point interface to Access Control API.
ACObject Class	Represents the accessControl GDMO object class.
ACRule Class	Represents the rule GDMO object class.
ACRuleContainer Class	Contains C++ Access Control Rule objects.
ACScope Class	Holds scope information. Convenience class.
ACTargets Class	Represents the targets GDMO object class.
ACTargetsContainer Class	Represents the targetContainer GDMO object class.
ACUser Class	Stores user information. Convenience class.

5.4 Symbolic Constants and Defined Types

5.4.1 Constants

ACAuxOwnerType

```
enum ACAuxOwnerType {  
    USER,  
    GROUP,  
    INVALID_OWNER_TYPE = -1  
}
```


ACAccessControlSwitch

```
enum ACAccessControlSwitch {  
    emAccessControlOff,  
    emAccessControlOn  
}
```

ACCallbackType

```
enum ACCallbackType {  
    OBJECT_CREATION_CALLBACK,  
    OBJECT_DELETION_CALLBACK,  
    ATTRIBUTE_VALUE_CHANGED_CALLBACK,  
    IMAGE_INCLUDED_CALLBACK,  
    IMAGE_EXCLUDED_CALLBACK,  
    RAW_EVENT_CALLBACK  
}
```

ACDenialGranularity

```
enum ACDenialGranularity {  
    request,  
    object,  
    attribute  
}
```

ACEMAuditLevel

```
enum ACEMAuditLevel  
{  
    AUDIT_OFF,  
    AUDIT_LEVEL1,  
    AUDIT_LEVEL2  
}
```

Symbolic Constants and Defined Types

ACEMSecurityLevel

```
enum ACEMSecurityLevel
{
    SECURITY_OFF,
    SECURITY_LEVEL1,
    SECURITY_LEVEL2
}
```

ACErrorType

```
enum ACErrorType {
    ACC_OK,
    ACC_FAILED,
    ACC_USER_EXISTS,
    ACC_USER_NOT_EXISTS,
    ACC_INVALID_USER_NAME,
    ACC_GROUP_EXISTS,
    ACC_GROUP_NOT_EXISTS,
    ACC_APPLICATION_EXISTS,
    ACC_APPLICATION_NOT_EXISTS,
    ACC_TARGETS_EXISTS,
    ACC_TARGETS_NOT_EXISTS,
    ACC_RULE_EXISTS,
    ACC_RULE_NOT_EXISTS,
    ACC_FEATURE_EXISTS,
    ACC_FEATURE_NOT_EXISTS,
    ACC_MOI_EXISTS,
    ACC_MOI_NOT_EXISTS,
    ACC_MOC_EXISTS,
    ACC_MOC_NOT_EXISTS,
    ACC_DB_OBJECT_TABLE_EXISTS,
    ACC_DB_OBJECT_TABLE_NOT_EXISTS,
    ACC_DB_OBJECT_ACCESS_EXISTS,
    ACC_DB_OBJECT_ACCESS_NOT_EXISTS,
    ACC_NO_MEMORY
}
```

ACObjectType

```
enum ACObjectType {  
    AC_TARGETS_OBJECT,  
    AC_RULE_OBJECT,  
    AC_APPLICATION_OBJECT,  
    AC_GROUP_OBJECT,  
    AC_DB_OBJECT,  
    AC_EM_NOTIFICATION_EMITTER  
}
```

ACTargetsType

```
enum ACTargetsType {  
    X741_TARGETS,  
    EM_TARGETS  
}
```

EnforcementAction

```
enum EnforcementAction {  
    denyWithResponse,  
    denyWithoutResponse,  
    abortAssociation,  
    denyWithFalseResponse,  
    allow  
}
```

5.4.2 Defined Types

This section lists the defined types.

ACAccessUserListSet

```
typedef RWTValSlist<ACUser> ACAccessUserListSet
```

ACApplicationAndFeatureList

```
typedef RWTValSlist<RWCString> ACApplicationAndFeatureList
```

ACApplicationFeatureList

```
typedef RWTValSlist<RWCString> ACApplicationFeatureList
```

ACApplicationList

```
typedef RWTValSlist<RWCString> ACApplicationList
```

ACDbObjectAccessList

```
typedef RWTValSlist<RWCString> ACDbObjectAccessList
```

ACDbObjectList

```
typedef RWTValSlist<RWCString> ACDbObjectList
```

ACDbObjectTableList

```
typedef RWTValSlist<RWCString> ACDbObjectTableList
```

ACDefaultAccess

```
typedef RWTValSlist<RWCString> ACDefaultAccess
```

ACDefaultEventAccess

```
typedef EnforcementAction ACDefaultEventAccess
```

ACDenialResponse

```
typedef EnforcementAction ACDenialResponse
```

ACDomainIdentity

```
typedef RWCString ACDomainIdentity
```

ACEventsDiscriminator

```
typedef RWCString ACEventsDiscriminator
```

ACFilter

```
typedef RWCString ACFilter
```

Symbolic Constants and Defined Types

ACGroupDescription

```
typedef RWCString ACGroupDescription
```

ACGroupList

```
typedef RWTValSlist<RWCString> ACGroupList
```

ACGroupMemberList

```
typedef RWTValSlist<RWCString> ACGroupMemberList
```

ACMOCList

```
typedef RWTValSlist<RWCString> ACMOCList
```

ACMOIList

```
typedef RWTValSlist<RWCString> ACMOIList
```

ACOperationsList

```
typedef RWTValSlist<RWCString> ACOperationsList
```

ACRuleList

```
typedef RWTValSlist<RWCString> ACRuleList
```

ACTargetsList

```
typedef RWTValSlist<RWCString> ACTargetsList
```

ACSuperUserList

```
typedef RWTValHashSet<RWCString> ACSuperUserList
```

ACTrustedHostList

```
typedef RWTValHashSet<RWCString> ACTrustedHostList
```

5.5 Access Control API Classes

This section describes the following Access Control API classes:

- `ACAccessControlRules`
- `ACAccessUserList`
- `ACAppFeatureContainer`
- `ACApplication`
- `ACApplicationContainer`
- `ACApplicationFeature`
- `ACCallback`
- `ACContainer`
- `ACDbObject`
- `ACDbObjectContainer`
- `ACEMNotificationEmitter`
- `ACEMTargets`
- `ACGroup`
- `ACGroupContainer`
- `ACInterface`
- `ACObject`
- `ACRule`
- `ACRuleContainer`
- `ACScope`
- `ACTargets`
- `ACTargetsContainer`
- `ACUser`

5.6 ACAccessControlRules Class

Inheritance: class ACOBJECT

```
#include <acapi/accesscontrolrules.hh>
```

Data Members: No public data members are declared in this class.

The `ACAccessControlRules` class represents the `emAccessControlRules` GDMO object class, which is defined in the Solstice EM Access Control module and is derived from X.741's `accessControlRules` GDMO object class.

The `emAccessControlRules` GDMO object class extends the `accessControlRules` GDMO object class by adding the following attributes:

- `accessControlSwitch`
- `trustedHostList`
- `defaultEventAccess`

The `ACAccessControlRules` class provides methods for accessing and modifying the default attribute values of the access control service. This class acts as a container for all the rules in the system.

5.6.1 Constructor

```
ACAccessControlRules()
```

The default constructor initializes the object that it represents, and prepares itself to register callbacks.

Note – It is possible to construct more than one `ACAccessControlRules` object. The additional objects, however, are references to the same object, because there can only be one instance of the `emAccessControlRules` GDMO object.

5.6.2 Destructor

```
~ACAccessControlRules()
```

5.6.3 ACAccessControlRules Member Functions

`add_trusted_hosts`

```
Result add_trusted_hosts(ACTrustedHostList& add_trusted_host_list)
```

Adds *add_trusted_host_list* to the list of trusted hosts. If a host is already included in the trusted hosts list, it is not added again.

`get_access_control_switch`

```
ACAccessControlSwitch get_access_control_switch()
```

Gets the access control status of the currently running Solstice EM. This function returns one of the following two values:

- `emAccessControlOff`

Indicates that any user can freely view, modify, or delete objects in the platform. Access control is not enforced.

- `emAccessControlOn`

Indicates that users need to be added to the platform and given appropriate privileges to view, modify, or delete objects. Access control is enforced.

ACAccessControlRules Class

get_default_access

```
ACDefaultAccess get_default_access()
```

Gets the default access value for each operation (action, create, delete, and so on).

Returns a list of value pairs of the form {<operation> <access>, <operation> <access>, ...}. For example:

```
{action denyWithResponse, create denyWithResponse, delete  
denyWithResponse, get denyWithResponse, replace denyWithResponse,  
addMember denyWithResponse, removeMember denyWithResponse,  
replaceWithDefault denyWithResponse, multipleObjectSelection  
denyWithResponse, filter denyWithResponse}
```

The default access value for all operations is denyWithResponse.

get_default_event_access

```
ACDefaultEventAccess get_default_event_access()
```

Gets the default action for events. It returns one of the following values:

- denyWithResponse (default)
- denyWithoutResponse
- abortAssociation
- denyWithFalseResponse
- allow

`get_denial_granularity`

```
ACDenialGranularity get_denial_granularity()
```

Returns one of the following three values that represent the level at which denial of access is exhibited:

- `request`

Access is denied at the *request* level. An entire request to access one or more managed objects in the MIS is denied if access to one of the managed objects in the request is denied. The request is allowed only when *all* managed objects in the request are accessible.

- `object`

Access is denied at the *object* level. Access is denied only to the request's managed objects that are not accessible. Access to the remaining managed objects in the request is allowed.

- `attribute`

Access is denied at the *attribute* level. Request to access a managed object is denied if access to one or more of its attributes is denied. Access to the managed object is allowed only when all the attributes of the managed object are accessible.

`get_denial_response`

```
ACDenialResponse get_denial_response()
```

Returns the denial response that access control sends out when denial is made because the default rule was satisfied.

The return values are as follows:

- `denyWithResponse` (default)
- `denyWithoutResponse`
- `abortAssociation`
- `denyWithFalseResponse`
- `allow`

ACAccessControlRules Class

get_domain_identity

```
ACDomainIdentity get_domain_identity()
```

Returns the access control domain identity that is governed by the access control rules. By default, the domain identity is EM.

get_trusted_host_list

```
ACTrustedHostList get_trusted_host_list(Boolean real) const
```

Returns the list of trusted hosts. Trusted hosts are systems that can freely connect as root to an MIS machine. The list of trusted hosts is maintained by the MIS server that holds the security profiles.

is_trusted_host

```
RWBoolean is_trusted_host(const RWCString& host_name) const
```

Checks whether *host_name* is in the list of trusted hosts. If *host_name* is part of the list, this function returns TRUE; otherwise, it returns FALSE.

remove_trusted_hosts

```
Result remove_trusted_hosts(ACTrustedHostList& remove_trusted_host_list)
```

Removes the hosts in *remove_trusted_host_list* from the list of trusted hosts. If a host in *remove_trusted_host_list* is not in the list of trusted hosts, it is ignored.

Returns TRUE on successful completion; otherwise, FALSE.

`replace_trusted_host_list`

```
Result replace_trusted_host_list(ACTrustedHostList&
new_trusted_host_list)
```

Replaces the trusted host list with *new_trusted_host_list*.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

`set_access_control_switch`

```
Result set_access_control_switch(ACAccessControlSwitch)
```

Sets the access control status in the MIS to one of the following values:

- `emAccessControlOff`

Any user can freely view, modify, or delete objects in the platform. Access control is not enforced.

- `emAccessControlOn`

Users need to be added to the platform and given appropriate privileges to view, modify, or delete objects. Access control is enforced.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

`set_default_access`

```
Result set_default_access(ACDefaultAccess&)
```

Sets the default access for each operation (action, create, delete, and so on) as a list of value pairs of the form {<operation> <access>, <operation> <access>, ...}.

For example:

```
{action denyWithResponse, create denyWithResponse, delete
denyWithResponse, get denyWithResponse, replace denyWithResponse,
addMember denyWithResponse, removeMember denyWithResponse,
replaceWithDefault denyWithResponse, multipleObjectSelection
denyWithResponse, filter denyWithResponse}
```

Returns TRUE on successful completion; otherwise, FALSE.

set_default_event_access

```
Result set_default_event_access(ACDefaultEventAccess)
```

Sets the access control status in the platform to one of the following values:

- denyWithResponse (default)
- denyWithoutResponse
- abortAssociation
- denyWithFalseResponse
- allow

Returns TRUE on successful completion; otherwise, FALSE.

set_denial_granularity

```
Result set_denial_granularity(ACDenialGranularity)
```

Sets the access denial level to one of the following values (passed through the *ACDenialGranularity* parameter):

- request

Access is denied at the *request* level. An entire request to access one or more managed objects in the MIS is denied if access to one of the managed objects in the request is denied. The request is allowed only when *all* managed objects in the request are accessible.

- `object`

Access is denied at the *object* level. Access is denied only to the request's managed objects that are not accessible. Access to the remaining managed objects in the request is allowed.

- `attribute`

Access is denied at the *attribute* level. Request to access a managed object is denied if access to one or more of its attributes is denied. Access to the managed object is allowed only when all the attributes of the managed object are accessible.

Note – Solstice EM only supports object-level access control.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

`set_denial_response`

```
Result set_denial_response(ACDenialResponse)
```

Sets the denial response to be returned by access control when the default rule is satisfied to one of the following values:

- `denyWithResponse` (default)
- `denyWithoutResponse`
- `abortAssociation`
- `denyWithFalseResponse`
- `allow`

Returns `TRUE` on successful completion; otherwise, `FALSE`.

5.7 ACAccessUserList Class

Inheritance: public ACOBJECT

```
#include <acapi/acaccessuserlist.hh>
```

Data Members: No public data members are declared in this class.

The `ACAccessUserList` class represents the `accessUserList` GDMO object in the MIS. This class stores the list of users registered under access control, and maintains a list of super users. Whenever any of the attribute values for this class changes, it sends `attributeValueChange` notifications.

5.7.1 Constructor

```
ACAccessUserList()
```

The default constructor initializes the `ACAccessUserList` class, and prepares it to register callbacks.

Note – It is possible to construct more than one `ACAccessUserList` object. The additional objects, however, are references to the same object, because there can only be one instance of the `accessUserList` GDMO object.

5.7.2 Destructor

```
~ACAccessUserList()
```


5.7.3 ACAccessUserList Member Functions

add_superusers

```
Result add_superusers(ACSuperUserList& add_superuser_list)
```

Adds *add_superuser_list* to the list of super users, unless a user in list is already a super user.

Returns TRUE on successful completion; otherwise, FALSE.

add_user

```
Result add_user(ACUser& user)
```

Adds *user* as a user under access control. If *user* already exists, the error type is set to ACC_USER_EXISTS, and the error string is set to “User exists in accessUserList!”.

Returns TRUE on successful completion; otherwise, FALSE.

get_access_user_list_set

```
ACAccessUserListSet get_access_user_list_set()
```

Returns a list of the users registered under access control.

get_superuser_list

```
ACSuperUserList get_superuser_list(Boolean real = TRUE) const
```

Returns a list of the super users under the access control domain.

ACAccessUserList Class

is_superuser

```
RWBoolean is_superuser(const RWCString& user_name) const
```

Returns TRUE if *user_name* is a super user; otherwise, FALSE.

replace_superuser_list

```
Result replace_superuser_list(ACSuperUserList& new_superuser_list)
```

Replaces the existing list of super users with *new_superuser_list*.

Returns TRUE on successful completion; otherwise, FALSE.

remove_superusers

```
Result remove_superusers(ACSuperUserList& remove_superuser_list)
```

Removes the super users that are specified in the *remove_superuser_list* list.

Returns TRUE on successful completion; otherwise, FALSE.

remove_user

```
Result remove_user(ACUser& user)
```

Removes *user* from the list of users under access control.

Returns TRUE on successful completion; otherwise, FALSE.

5.8 ACAppFeatureContainer Class

Inheritance: class ACContainer

```
#include <acapi/acapplicationfeature.hh>
```

Data Members: No public data members are declared in this class.

The ACAppFeatureContainer class is a container for all the features that can be controlled through Solstice EM's feature-level access control for a given application.

5.8.1 Constructor

```
ACAppFeatureContainer(const RWCString& appl_name)
```

The constructor creates a feature container object for *appl_name*. If an object has already been created for *appl_name*, the object is not created again. Instead, the object's internal reference count is incremented by one.

5.8.2 Destructor

```
~ACAppFeatureContainer()
```

5.8.3 ACAppFeatureContainer Member Functions

get_all_features

```
ACApplicationFeatureList get_all_features()
```

Returns all the features that have been registered for feature-level access control for the application that this class represents.

ACApplication Class

get_container_name

```
RWCString get_container_name()
```

Returns the container name, which is the application name.

get_feature

```
ACApplicationFeature get_feature(const RWCString& featurename)
```

Returns the ACApplicationFeature object *featurename*.

5.9 ACApplication Class

Inheritance: class ACOBJECT

```
#include <acapi/acapplication.hh>
```

Data Members: No public data members are declared in this class.

The ACApplication class represents the application GDMO object class defined in the Solstice EM Access Control module. This object is a container for all the features that are controlled through Solstice EM's feature-level access control for a given application.

5.9.1 Constructor

```
ACApplication(const RWCString& appl_name)
```

The constructor creates an application object whose name is the value of *appl_name*. If an object has already been created for *appl_name*, the object is not created again. Instead, the object's internal reference count is incremented by one.

5.9.2 Destructor

```
~ACApplication()
```

5.9.3 ACApplication Member Functions

`destroy`

```
Result destroy()
```

Removes the application object that this class represents from all groups to which it belongs. In addition, deletes the application object and its features from access control. Upon the deletion, the application is no longer subject to access control.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

`get_application_description`

```
RWCString get_application_description()
```

Gets the application description information.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

`set_application_description`

```
Result set_application_description(const RWCString& desc)
```

Sets *desc* as the application's description information.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

5.10 ACApplicationContainer Class

Inheritance: public ACContainer

```
#include <acapi/acapplication.hh>
```

Data Members: No public data members are declared in this class.

The ACApplicationContainer class represents the applicationContainer GDMO object class which is defined in the Solstice EM Access Control module. This object is a container for all the applications that can be controlled through Solstice EM's feature-level access control.

5.10.1 Constructor

```
ACApplicationContainer()
```

The constructor creates an application container object that contains all the applications that are subject to access control. If such an object has already been created, its internal reference count is incremented by one.

5.10.2 Destructor

```
~ACApplicationContainer()
```

5.10.3 ACApplicationContainer Member Functions

get_all_applications

```
ACApplicationList get_all_applications()
```

Returns a list of all the applications under the `ACApplicationContainer` object which, by default, includes most of the Solstice EM applications.

`get_application`

```
ACApplication get_application(const RWCString& appname)
```

Returns the `ACApplication` object whose name is stored in *appname*.

5.11 ACApplicationFeature Class

Inheritance: public `ACObject`

```
#include <acapi/acapplicationfeature.hh>
```

Data Members: No public data members are declared in this class.

The `ACApplicationFeature` class represents the `applicationFeature` GDMO object class which is defined in the Solstice EM Access Control module. `ACApplicationFeature` represents a feature that can be controlled through Solstice EM's feature-level access control for a given application.

5.11.1 Constructor

```
ACApplicationFeature(const RWCString& appname, const RWCString& featurename)
```

The constructor creates an `applicationFeature` object based on the values of *appname* and *featurename*. If such an object already exists, the internal reference count is incremented by one, and the object is returned.

5.11.2 Destructor

```
~ACApplicationFeature()
```

5.11.3 ACApplicationFeature Member Functions

destroy

```
Result destroy()
```

Removes an application's feature object from all groups to which it belongs and deletes the feature object from access control. Upon deletion, the application's feature is no longer subject to access control.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

get_feature_description

```
RWCString get_feature_description()
```

Returns the feature's description.

set_feature_description

```
Result set_feature_description(const RWCString& desc)
```

Sets *desc* as the description for the feature.

Returns `TRUE` if successful; otherwise, `FALSE`.

5.12 ACCallback Class

Inheritance: class Callback

```
#include <acapi/accallback.hh>
```

This class is a simple extension of the PMI callback class.

5.12.1 Constructors

```
ACCallback()
```

The default constructor initializes the Callback object.

```
ACCallback(CallbackHandler hand, Ptr userdata, ACCallbackType type)
```

In the preceding constructor:

- *hand*, defined in `pmi/callback.hh`, is a pointer to the function that the scheduler must call.
- *type* can be one of the following:
 - OBJECT_CREATION_CALLBACK
 - OBJECT_DELETION_CALLBACK
 - ATTRIBUTE_VALUE_CHANGED_CALLBACK
 - IMAGE_INCLUDED_CALLBACK
 - IMAGE_EXCLUDED_CALLBACK
 - RAW_EVENT_CALLBACK

```
ACCallback(const ACCallback& other)
```

The preceding is a copy constructor.

5.12.2 Destructor

```
~ACCallback()
```

5.12.3 ACCallback Operator Overloading

```
ACCallback& operator = (const ACCallback& other)
```

The assignment operator works like the copy constructor.

5.12.4 ACCallback Member Functions

`exec_callback`

```
void exec_callback(Ptr call_data) const
```

Invokes the callback's handler with the callback's data, and `exec`'s *call_data* as arguments.

`get_callback_type`

```
ACCallbackType get_callback_type() const
```

Returns the callback's type.

5.13 ACContainer Class

Inheritance: None

```
#include <acapi/accontainer.hh>
```

Data Members: No public data members are declared in this class.

The `ACContainer` class is an abstract base class that abstracts the concept of a container for access control GDMO objects.

This class is subclassed to derive more specific classes that serve as an interface to containers of various X.741 GDMO object classes. For example, the subclass `ACRuleContainer` C++ provides the interface to the container of X.741 rule GDMO objects.

Multiple `ACContainer` objects that represent the same container of X.741 GDMO objects share the same object by maintaining a reference count. The `ACContainer` class allows its subclasses to register any object inclusion and object exclusion events from the container and any raw events from PMI, so that the container is updated dynamically and automatically.

5.13.1 Constructor

This constructor is protected so that the user cannot instantiate this object.

5.13.2 Destructor

```
virtual ~ACContainer() = 0;
```

5.13.3 ACContainer Operator Overloading

```
Boolean operator == (const ACContainer& self, const ACContainer&
other)
```

If the two compared container objects have the same object names, the preceding comparison operator returns TRUE.

```
ACContainer& operator=(const ACContainer&)
```

The preceding operator overloads the assignment operator.

5.13.4 ACContainer Member Functions

add_callback

```
void add_callback(const ACCallback& cb)
```

Adds the given callback pointer to ACContainer's callback queue.

Note – You can add more than one callback for a given type of event. This allows you to use multiple callbacks to process the same event.

The following six types of events are supported:

- OBJECT_CREATION
- OBJECT_DELETION
- ATTRIBUTE_VALUE_CHANGED
- IMAGE_INCLUDED
- IMAGE_EXCLUDED
- RAW_EVENT

`get_error_string`

```
RWCString get_error_string()
```

Returns the error string stored in the `ACAccessControl` object that pertains to the object that this function represents.

`get_error_type`

```
ACErrorType get_error_type()
```

Returns the error type stored in the `ACAccessControl` object that pertains to the object that this function represents. The possible values of the error type are:

- `ACC_APPLICATION_EXISTS`
- `ACC_APPLICATION_NOT_EXISTS`
- `ACC_DB_OBJECT_ACCESS_EXISTS`
- `ACC_DB_OBJECT_ACCESS_NOT_EXISTS`
- `ACC_DB_OBJECT_TABLE_EXISTS`
- `ACC_DB_OBJECT_TABLE_NOT_EXISTS`
- `ACC_FAILED`
- `ACC_FEATURE_EXISTS`
- `ACC_FEATURE_NOT_EXISTS`
- `ACC_GROUP_EXISTS`
- `ACC_GROUP_NOT_EXISTS`
- `ACC_INVALID_USER_NAME`
- `ACC_MOC_EXISTS`
- `ACC_MOC_NOT_EXISTS`
- `ACC_MOI_EXISTS`
- `ACC_MOI_NOT_EXISTS`
- `ACC_NO_MEMORY`
- `ACC_OK`
- `ACC_RULE_EXISTS`
- `ACC_RULE_NOT_EXISTS`
- `ACC_TARGETS_EXISTS`
- `ACC_TARGETS_NOT_EXISTS`
- `ACC_USER_EXISTS`
- `ACC_USER_NOT_EXISTS`

`get_name_only`

```
RWCString get_name_only()
```

Returns the name of the `ACAccessControl` object that pertains to the `ACContainer`.

ACContainer Class

get_object_name

```
RWCString get_object_name() const
```

Returns the name of the `ACAccessControl` object stored in the distinguished name (DN) format.

remove_callback

```
void remove_callback(const ACCallback& cb)
```

Removes the callback associated with its object.

reset_error

```
void reset_error()
```

Resets the error state of the `ACAccessControl` object that pertains to the `ACContainer` by setting both the error string and error type to `ACC_OK`.

set_error

```
void set_error(ACErrorType type, const RWCString& err)
```

Sets the error type to the *type* argument, and the error string to the *err* argument. If *type* is `ACC_OK`, this method performs the same action as the `reset_error` method.

set_error_string

```
void set_error_string(const RWCString& err)
```

Sets the error string to the *err*.

set_error_type

```
void set_error_type(ACErrorType type)
```

Sets the error type to the *type* argument. If *type* is ACC_OK, this method performs the same action as the reset_error method.

5.14 ACDbObject Class

Inheritance: public ACOBJECT

```
#include <acapi/acdbobject.hh>
```

Data Members: No public data members are declared in this class.

The ACDbObject class represents the emDbObject GDMO object class from the Solstice EM DB Info module. An emDbObject object represents a database object on which access control can be specified.

5.14.1 Constructor

```
ACDbObject(const RWCString& objectname)
```

The constructor creates an emDbObject object whose name is specified by *objectname*. If a Dbobject with the same name already exists, the internal reference count is incremented by one and the object is returned.

5.14.2 Destructor

```
~ACDbObject()
```

5.14.3 ACDBObject Member Functions

add_db_object_access

```
Result add_db_object_access(const RWCString& access)
```

Adds the given argument to the `emDBObject` object's access list.

Returns `TRUE` on successful completion. Otherwise, if `access` already exists in the access list, this function sets the error type to `ACC_DB_OBJECT_ACCESS_EXISTS` and the error string to "Access exists in `emDBObjectAccessList!`", and returns `FALSE`.

add_db_object_table

```
Result add_db_object_table(const RWCString& table)
```

Adds `table` to `emDBObject`'s table list.

Returns `TRUE` on successful completion. Otherwise, if `table` already exists in the table list, this function sets the error type to `ACC_DB_OBJECT_ACCESS_EXISTS` and the error string to "Access exists in `emDBObjectAccessList!`", and returns `FALSE`.

get_db_object_access_list

```
ACDBObjectAccessList get_db_object_access_list()
```

Returns the access list that contains the names of the groups that can access `DBObject`.

get_db_object_table_list

```
ACDBObjectTableList get_db_object_table_list()
```

Returns the list of tables that are under access control for the `DBObject` object.

`remove_db_object_access`

```
Result remove_db_object_access(const RWCString& access)
```

Removes *access* from the DbObject object's access list.

Returns `TRUE` on successful completion. Otherwise, if *access* does *not* exist, this function sets the error type to `ACC_DB_OBJECT_ACCESS_NOT_EXISTS` and the error string to "Access doesn't exist in emDBObjectAccessList!", and returns `FALSE`.

`remove_db_object_table`

```
Result remove_db_object_table(const RWCString& table)
```

Removes *table* from the DbObject object's table list.

Returns `TRUE` on successful completion. Otherwise, if *table* does not exist, this function sets the error type to `ACC_DB_OBJECT_TABLE_NOT_EXISTS` and the error string to "Table doesn't exist in emDBObjectTableList!", and returns `FALSE`.

`set_db_object_access_list`

```
Result set_db_object_access_list(ACDBObjectAccessList& accesslist)
```

Replaces DbObject's current access with *accesslist*.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

`set_db_object_table_list`

```
Result set_db_object_table_list(ACDBObjectTableList& tablelist)
```

Replaces the current table list of the DbObject with *tablelist*.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

ACDBObject Class

set_auxobject_owner

```
Result set_auxobject_owner(   ACAuxOwnerType owntype,  
                             RWCString      ownid)
```

The above method sets the owner of the auxiliary object to *ownid* and the type of the owner to *owntype*. Returns `TRUE` on successful completion; otherwise, `FALSE`.

get_auxobject_owner_type

```
ACAuxOwnerType get_auxobject_owner_type()
```

The above method returns the type of owner, that is `USER` or `GROUP` of the auxiliary object. If the owner type is not known or incorrect `INVALID_OWNER_TYPE` is returned.

get_auxobject_owner_id

```
RWCString get_auxobject_owner_id()
```

The above method returns the owner id of the auxiliary object, this is valid only when the owner type of the auxiliary object is `USER` or `GROUP`.

5.14.4 Notes About the ACDBObject Class

The default owner of an auxiliary object is created as root. In order to change the owner, you need to set the owner id using the `set_auxobject_owner()` method. The owner type can be `USER` or `GROUP` and the owner id can be a the name of a user or the name of a group.

After setting the auxiliary object's owner, you must perform a `create()` or `store()` operation so that the changes in the auxiliary object are pushed down to the MIS. CODE EXAMPLE 5-1 illustrates the sequence for performing a `create()` operation (M-CREATE).

CODE EXAMPLE 5-1 Sequence for Performing a `create()` Operation

```
// Create the ACDBObject: Create an auxiliary object with root as
the
// default owner id
    ACDBObject *acdbobj_ptr = new ACDBObject(logname, FALSE);
    // Set the auxiliary objects owner
    if (!acdbobj_ptr->set_auxobject_owner(USER,
owner_id_str.chp())) {
        cout << "Failed to set the owner id." << endl;
        exit(0);
    }
    // Create the acdbobject and its auxiliary object
    if (!acdbobj_ptr->create()) {
        cout << "Failed to create the acdbobject." << endl;
        exit(1);
    }
The sequence for changing the owner of a log is as follows: (M-SET)
    // Set the auxiliary objects owner
    //-----
    if (!acdbobj_ptr->set_auxobject_owner(USER,
owner_id_str.chp())) {
        cout << "Failed to set the owner id." << endl;
        exit(0);
    }
    // Store the changes to the owner of the auxiliary object
    if (!acdbobj_ptr->store_auxobject()) {
        cout << "Failed to change the auxiliary objects owner." <<
endl;
        exit(1);
    }
```

Note – Two test programs that show how to create and set an `ACDBObject` object are supplied in the `/opt/SUNWconn/em/src/ac_api` directory.

5.15 ACDbObjectContainer Class

Inheritance: public ACContainer

```
#include <acapi/acdbobject.hh>
```

Data Members: No public data members are declared in this class.

The ACDbObjectContainer class represents the emDbInfo GDMO object class from the Solstice EM DB Info module. ACDbObjectContainer is a container for all ACDbObject objects. It stores database-specific access control information.

5.15.1 Constructor

```
ACDbObjectContainer()
```

The default constructor creates the container object. If the object has already been created, the internal reference count for the object is incremented by one.

5.15.2 Destructor

```
~ACDbObjectContainer()
```

5.15.3 ACDbObjectContainer Member Functions

get_access_db_objects

```
ACDbObjectList get_access_db_objects(const RWCString& group)
```

Returns all ACDbObject objects that contain the given group in the ACDbObjectContainer object's access list.

get_all_db_objects

```
ACDBObjectList get_all_db_objects()
```

Returns all ACDBObject objects that are stored in ACDBObjectContainer.

get_db_object

```
ACDBObject get_db_object(const RWCString& appname)
```

Returns the ACDBObject object whose name is specified by *appname*.

get_db_server_name

```
RWCString get_db_server_name()
```

Returns the database server name.

get_db_server_type

```
RWCString get_db_server_type()
```

Returns the database server type.

5.16 ACEMNotificationEmitter Class

Inheritance: public ACOBJECT

```
#include <acapi/notificationemitter.hh>
```

Data Members: No public data members are declared in this class.

The `ACEMNotificationEmitter` class represents the `emNotificationEmitter` GDMO object class from the Solstice EM Access Control module. The `emNotificationEmitter` object represents a notification emitter for security alarm and auditing.

5.16.1 Constructor

```
ACEMNotificationEmitter()
```

The constructor creates a `notificationEmitter` object. If such an object has already been created, the internal reference count for the object is incremented by one.

5.16.2 Destructor

```
~ACEMNotificationEmitter()
```

5.16.3 ACEMNotificationEmitter Member Functions

get_audit_level

```
ACEMAuditLevel get_audit_level()
```

Returns one of the following values that represent the audit level:

- AUDIT_OFF
- AUDIT_LEVEL1
- AUDIT_LEVEL2

get_invalid_access_attempts

```
long get_invalid_access_attempts()
```

Returns a count of the number of times that access was denied.

get_security_level

```
ACEMSecurityLevel get_security_level()
```

Returns one of the following values that represent the security level:

- SECURITY_OFF
- SECURITY_LEVEL1
- SECURITY_LEVEL2

get_valid_access_attempts

```
long get_valid_access_attempts()
```

Returns a count of the number of times that an access control decision function authorized access.

ACEMTargets Class

set_audit_level

```
Result set_audit_level(ACEMAuditLevel)
```

Sets the audit level to one of the following values:

- AUDIT_OFF
- AUDIT_LEVEL1
- AUDIT_LEVEL2

Returns TRUE on successful completion; otherwise, FALSE.

set_security_level

```
Result set_security_level(ACEMSecurityLevel)
```

Sets the security level to one of the following values:

- SECURITY_OFF
- SECURITY_LEVEL1
- SECURITY_LEVEL2

Returns TRUE on successful completion; otherwise, FALSE.

5.17 ACEMTargets Class

Inheritance: public ACTargets

```
#include <acapi/acemtargets.hh>
```

Data Members: No public data members are declared in this class.

The ACEMTargets class represents the emTargets GDMO object class which is defined in the Solstice EM Access Control module. emTargets is derived from X.741's targets GDMO object class. ACEMTargets class adds an eventDiscriminator attribute to the targets class to be used for event access control.

5.17.1 Constructor

```
ACEMTargets(const RWCString& objname)
```

The preceding constructor creates an `ACEMTargets` object whose name is specified by *objname*. If an object has already been created with the same name, the internal reference count for the object is incremented by one.

5.17.2 Destructor

```
~ACEMTargets()
```

5.17.3 ACEMTargets Member Functions

`get_event_discriminator`

```
ACEEventsDiscriminator get_event_discriminator()
```

Returns the `ACEMTargets` object's event discriminator.

`set_event_discriminator`

```
Result set_event_discriminator(ACEEventsDiscriminator)
```

Sets the `ACEMTargets` object's event discriminator to *ACEEventsDiscriminator*.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

5.18 ACGroup Class

Inheritance: public ACOBJECT

```
#include <acapi/acgroup.hh>
```

Data Members: No public data members are declared in this class.

The ACGroup class represents the group GDMO object class which is defined in the Solstice EM Access Control module. group is derived from X.741's aclInitiators GDMO object class.

5.18.1 Constructor

```
ACGroup(const RWCString& objectname)
```

The constructor creates an ACGroup object whose name is specified by *objectname*. If an object with the same name has already been created, the internal reference count for the object is incremented by one.

5.18.2 Destructor

```
~ACGroup( )
```

5.18.3 ACGroup Member Functions

add_application

```
Result add_application(const RWCString& app)
```

Adds *app* to the list of applications that can be accessed by its group.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

add_application_feature

```
Result add_application_feature(const RWCString& app, const RWCString&  
feature)
```

Adds *feature* to the feature list of *app* that can be accessed by its group.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

add_group_member

```
Result add_group_member(const RWCString& membername)
```

Adds the given member to its group.

Returns `TRUE` on successful completion. Otherwise, if the given member already exists in the group, this function sets the error type to `ACC_USER_EXISTS` and the error string to “User exists in groupMemberList!”, and returns `FALSE`.

destroy

```
Result destroy()
```

Removes the instantiated group object from all containers that contains it, and then removes the object itself.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

ACGroup Class

get_all_applications_full_access

```
Boolean get_all_applications_full_access()
```

Returns `TRUE` if the group it represents contains the DN of the application container in its `applicationAndFeatureList`, which indicates that the group has full access to all applications; otherwise, `FALSE`.

A group has full access to all applications in an application container if the group contains the DN of the application container. This method returns `TRUE` if the instantiated group contains the DN of an application container in the `applicationAndFeatureList` of the group; otherwise, `FALSE`.

get_applications

```
ACApplicationList get_applications()
```

Returns all the applications that are accessible by its group.

get_application_and_feature_list

```
ACApplicationAndFeatureList  
get_application_and_feature_list(Boolean real)
```

If `real` is `TRUE`, this function returns a list of the DNs of all MIS applications and features that are accessible by the instantiated group; otherwise, it returns a list of the DNs of all applications and features in its own application space.

get_application_feature

```
Result add_application_feature(const RWCString& app, const  
RWCString& feature)
```

Adds *feature* to the feature list of *<app>* that can be accessed by its group.

Returns `TRUE` on successful completion; otherwise, `FALSE`. For more information, check the error type and string by calling the `get_error_type` function and the `get_error_string` function. Verify that the application has full access by calling `get_application_full_access`.

`get_application_features`

```
ACApplicationFeatureList get_application_features(
    const RWCString& appname)
```

This function returns all the features of the given application that are accessible by its group. If the given application has full access, this function returns an empty list. Verify that the application has full access by calling `get_application_full_access`.

`get_application_full_access`

```
Boolean get_application_full_access(const RWCString& appname)
```

Checks the `applicationAndFeatureList` of the instantiated group for the DN of the given application, to determine whether the group has full access to the given application.

Returns `TRUE` if its group has full access to the given application; otherwise, `FALSE`.

`get_group_description`

```
ACGroupDescription get_group_description()
```

Returns the `ACGroup` object's description information.

`get_group_member_list`

```
ACGroupMemberList get_group_member_list()
```

Returns a list of the group members (users).

ACGroup Class

remove_application

```
Result remove_application(const RWCString& app)
```

Removes the given application and all of its associated features from the applicationAndFeatureList of the group.

Returns TRUE on successful completion; otherwise, FALSE.

remove_application_feature

```
Result remove_application_feature(const RWCString& app, const  
RWCString& feature)
```

Removes the given feature from the given application.

Returns TRUE on successful completion; otherwise, FALSE.

If the given feature does not exist, this function sets the error type to ACC_FEATURE_NOT_EXISTS and the error string to "Application feature doesn't exist in applicationAndFeatureList!".

remove_group_member

```
Result remove_group_member(const RWCString& membername)
```

Removes the given member (user) from its groupMemberList.

Returns TRUE on successful completion; otherwise, FALSE.

If the given member does not exist, this function sets the error type to ACC_USER_NOT_EXISTS and the error string to "User doesn't exist in groupMemberList!".

set_all_applications_full_access

```
Result set_all_applications_full_access(Boolean fullaccess)
```

A group has full access to all applications in an application container if the group contains the DN of the application container.

If *fullaccess* is `TRUE`, this method adds the DN of an application container to the `applicationAndFeatureList` of the instantiated group.

If *fullaccess* is `FALSE`, this method removes the DN from the `applicationAndFeatureList` of the instantiated group.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

`set_application_and_feature_list`

```
Result set_application_and_feature_list(A
                                     CApplicationAndFeatureList& list)
```

Sets the given list as the `applicationAndFeatureList` for the instantiated group.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

`set_application_full_access`

```
Result set_application_full_access(const RWCString& application)
```

Sets the DN of the given application in the `applicationAndFeatureList` of the instantiated group so that the group has full access to the given application.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

`set_group_description`

```
Result set_group_description(const ACGroupDescription& description)
```

Sets the description of the instantiated group object to *description*.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

ACGroupContainer Class

set_group_member_list

```
Result set_group_member_list(ACGroupMemberList& list)
```

Sets the member (user) list of the instantiated group to *list*.

Returns TRUE on successful completion; otherwise, FALSE.

get_initiator_aci_mandated

```
Boolean get_initiator_aci_mandated()
```

Returns TRUE if the initiator of the instantiated ACGroup object is authorized; otherwise, FALSE.

set_initiator_aci_mandated

```
Result set_initiator_aci_mandated(Boolean mandated)
```

If *mandated* is TRUE, sets the initiator of the instantiated ACGroup object to authorized; otherwise, sets the initiator to unauthorized.

Returns TRUE on successful completion; otherwise, FALSE.

5.19 ACGroupContainer Class

Inheritance: public ACContainer

```
#include <acapi/acgroup.hh>
```

Data Members: No public data members are declared in this class.

The ACGroupContainer class represents the groupContainer GDMO object class which is defined in the Solstice EM Access Control module. groupContainer is a container for all the groups that are defined under access control of Solstice EM.

5.19.1 Constructor

```
ACGroupContainer()
```

The constructor creates a group container object that, when instantiated, contains all the groups that are subject to access control. By default, the following groups are available: full access, operator, and view-only. If such an object exists, the internal reference count for the object is incremented by one.

5.19.2 Destructor

```
~ACGroupContainer()
```

5.19.3 ACGroupContainer Member Functions

`get_all_groups`

```
ACGroupList get_all_groups()
```

Returns all the groups under the `groupContainer` object in the form of a group list.

`get_group`

```
ACGroup get_group(const RWCString& groupname)
```

Returns the group object with the given name.

ACInterface Class

get_user_group_list

```
ACGroupList get_user_group_list(const RWCString& username)
```

Returns all the groups containing a user with the given name.

5.20 ACInterface Class

Inheritance: None

```
#include <acapi/acinterface.hh>
```

Data Members: No public data members are declared in this class.

The ACInterface class is a convenience class that you can use to get all the container objects for access control and their contained objects.

5.20.1 Constructor

```
ACInterface()
```

The default constructor does nothing because there are no data members to initialize.

5.20.2 Destructor

```
~ACInterface()
```

5.20.3 ACInterface Member Functions

`get_access_user_list`

```
ACAccessUserList get_access_user_list();
```

Creates an `ACAccessUserList` object if it does not exist and returns it.

`get_application_container`

```
ACApplicationContainer get_application_container()
```

Creates an `ACApplicationContainer` object if it does not exist and returns it.

`get_db_object_container`

```
ACDBObjectContainer get_db_object_container()
```

Creates an `ACDBObjectContainer` object if it does not exist and returns it.

`get_em_notification_emitter`

```
ACEMNotificationEmitter get_em_notification_emitter()
```

Creates an `ACEMNotificationEmitter` object if it does not exist and returns it. If such an object exists, its internal reference count is incremented by one.

ACInterface Class

get_feature_container

```
ACAppFeatureContainer get_feature_container(const RWCString& appname)
```

Creates an ACAppFeatureContainer object (for the given application) if it does not exist and returns it.

get_group_container

```
ACGroupContainer get_group_container()
```

Creates an ACGroupContainer object if it does not exist and returns it.

get_rule_container

```
ACRuleContainer get_rule_container()
```

Creates an ACRuleContainer object if it does not exist and returns it.

get_targets_container

```
ACTargetsContainer get_targets_container()
```

Creates an ACTargetsContainer object if it does not exist and returns it.

5.21 ACObject Class

Inheritance: None

```
#include <acapi/acobject.hh>
```

Data Members: No public data members are declared in this class.

The `ACObject` class represents the `accessControl` managed object class, an abstract base class, as defined in X.741. `ACObject` is subclassed to derive more specific classes that serve as an interface to X.741 GDMO object classes.

`ACObject` should be used as a base class for only those classes that represent a concrete GDMO object class; that is, one that can be instantiated. For example, the `ACRule C++` class derives from the `ACObject` class and provides the interface to the X.741 rule GDMO object class.

5.21.1 Constructor

There is no default public constructor available for this class.

```
ACObject(const ACObject&)
```

A copy constructor is defined whose declaration is shown above.

5.21.2 Destructor

```
virtual ~ACObject() = 0
```

When the reference count for this object becomes 0, the object is deleted.

5.21.3 ACObject Operator Overloading

```
ACObject& operator = (const ACObject&)
```

The preceding is the declaration of an assignment operator.

```
Boolean operator == (const ACObject& other)
```

The preceding comparison operator returns TRUE if the objects on both sides are the same; otherwise, FALSE.

5.21.4 ACObject Member Functions

`add_callback`

```
void add_callback(const ACCallback& cb)
```

Adds the given callback to receive one of the following three events:

- `objectCreation`
- `objectDeletion`

It is your responsibility to delete the object name to free the allocated memory for object name.

- `attributeValueChange`

As part of the call data to the callback, the instantiated ACObject object is passed.

Note – It is possible to add more than one callback for the same event type.

copy

```
Result copy(ACObject& source)
```

Makes a copy of the given ACObject.

Returns TRUE on successful completion; otherwise, FALSE.

create

```
Result create()
```

Creates an ACObject object that represents the accessControl GDMO object. This is analogous to the `Image::create()` function in PMI.

Returns TRUE on successful completion; otherwise, FALSE. .

destroy

```
virtual Result destroy()
```

Destroys an ACObject that represents the accessControl GDMO object. This is analogous to the `Image::destroy()` function in PMI.

Returns TRUE on successful completion; otherwise, FALSE.

exists

```
Boolean exists()
```

Checks whether the object exists. This is analogous to the `Image::exists()` function in PMI.

Returns TRUE on successful completion; otherwise, FALSE.

ACObject Class

get_error_string

```
RWCString get_error_string() const
```

Returns the error string stored in the `ACAccessControl` object that pertains to the instantiated `ACObject`.

get_error_type

```
ACErrorType get_error_type() const;
```

Returns the error type stored in the `ACAccessControl` object that pertains to the instantiated `ACObject`. The error type can have one of the following values:

- | | |
|--|---------------------------------------|
| • <code>ACC_APPLICATION_EXISTS</code> | • <code>ACC_MOC_EXISTS</code> |
| • <code>ACC_APPLICATION_NOT_EXISTS</code> | • <code>ACC_MOC_NOT_EXISTS</code> |
| • <code>ACC_DB_OBJECT_ACCESS_EXISTS</code> | • <code>ACC_MOI_EXISTS</code> |
| • <code>ACC_DB_OBJECT_ACCESS_NOT_EXISTS</code> | • <code>ACC_MOI_NOT_EXISTS</code> |
| • <code>ACC_DB_OBJECT_TABLE_EXISTS</code> | • <code>ACC_NO_MEMORY</code> |
| • <code>ACC_DB_OBJECT_TABLE_NOT_EXISTS</code> | • <code>ACC_OK</code> |
| • <code>ACC_FAILED</code> | • <code>ACC_RULE_EXISTS</code> |
| • <code>ACC_FEATURE_EXISTS</code> | • <code>ACC_RULE_NOT_EXISTS</code> |
| • <code>ACC_FEATURE_NOT_EXISTS</code> | • <code>ACC_TARGETS_EXISTS</code> |
| • <code>ACC_GROUP_EXISTS</code> | • <code>ACC_TARGETS_NOT_EXISTS</code> |
| • <code>ACC_GROUP_NOT_EXISTS</code> | • <code>ACC_USER_EXISTS</code> |
| • <code>ACC_INVALID_USER_NAME</code> | • <code>ACC_USER_NOT_EXISTS</code> |

get_name_only

```
RWCString get_name_only() const
```

Returns the name of the instantiated `ACObject`.

`get_object_name`

```
RWCString get_object_name() const
```

Returns the name of the instantiated `ACObject` in the distinguished name (DN) format.

`remove_callback`

```
void remove_callback(const ACCallback& cb)
```

Removes the given callback.

`reset_error`

```
void reset_error()
```

Resets the error state of the object by setting the error string to “`ACC_OK`” and the error type to `ACC_OK`.

`revert`

```
Result revert()
```

Reverts the state of the object by canceling any pending set operation that has not yet been stored.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

`set_error`

```
void set_error(ACErrorType type, const RWCString& err)
```

Sets the error type to the given type and error string to the given string. A type value of `ACC_OK` performs a `reset_error` on the object.

ACRule Class

set_error_string

```
void set_error_string(const RWCString& err)
```

Sets the error string to the given string.

set_error_type

```
void set_error_type(ACErrorType type)
```

Sets the error type to the given type. A `type` value of `ACC_OK` performs a `reset_error` on the object.

store

```
Result store()
```

Stores the object that will be the representation of the GDMO object. This is analogous to the `Image::store()` function in PMI.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

5.22 ACRule Class

Inheritance: `public ACOBJECT`

```
#include <acapi/acrule.hh>
```

Data Members: No public data members are declared in this class.

The `ACRule` class represents X.741's `rule` GDMO object class, which grants or denies access. If the value of the enforcement action attribute is `allow`, access is permitted. Otherwise, the enforcement action attribute defines the type of denial response made to the initiator of the management operation.

5.22.1 Constructor

```
ACRule(const RWCString& objectname);
```

The constructor creates an `ACRule` object whose name is specified by `objectname`. If such an object with this name has already been created, the internal reference count for the object is incremented by one.

5.22.2 Destructor

```
~ACRule()
```

5.22.3 ACRule Member Functions

`add_group`

```
Result add_group(const RWCString& groupname)
```

Adds the given group to the rule's initiator list.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

If the given group already exists, this function sets the error type to `ACC_GROUP_EXISTS` and the error string to "Group exists in initiatorsList!".

ACRule Class

add_targets

```
Result add_targets(const RWCString& target)
```

Adds the given target to the rule's target list.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

If the given target already exists, this function sets the error type to `ACC_TARGETS_EXISTS` and the error string to "Targets exists in initiatorsList!".

get_enforcement_action

```
EnforcementAction get_enforcement_action()
```

Returns the enforcement action that is defined for this rule, which can be one of the following:

- `denyWithResponse`
- `denyWithoutResponse`
- `abortAssociation`
- `denyWithFalseResponse`
- `allow`

get_group_list

```
ACGroupList get_group_list()
```

Returns a list of the groups that belong to the rule object.

get_targets_list

```
ACTargetsList get_targets_list()
```

Returns a list of the targets that belong to the rule object.

`remove_group`

```
Result remove_group(const RWCString& groupname)
```

Removes the given group from the rule's initiator's list.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

If the given group already exists, this function sets the error type to `ACC_GROUP_NOT_EXISTS` and the error string to "Group doesn't exist in initiatorsList!".

`remove_targets`

```
Result remove_targets(const RWCString& target)
```

Removes the given target from the rule's target list.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

If the given target already exists, this function sets the error type to `ACC_TARGETS_NOT_EXISTS` and the error string to "Targets doesn't exist in initiatorsList!".

`set_enforcement_action`

```
Result set_enforcement_action(EnforcementAction action)
```

Sets the given action as the enforcement action for the rule. The given action can be one of the following:

- `denyWithResponse`
- `denyWithoutResponse`
- `abortAssociation`
- `denyWithFalseResponse`
- `allow`

Returns `TRUE` on successful completion; otherwise, `FALSE`.

ACRuleContainer Class

set_group_list

```
Result set_group_list(ACGroupList& grouplist)
```

Sets the given list as the rule's initiator list.

Returns TRUE on successful completion; otherwise, FALSE.

set_targets_list

```
Result set_targets_list(ACTargetsList& targetlist)
```

Sets the given list as the rule's target list.

Returns TRUE on successful completion; otherwise, FALSE.

5.23 ACRuleContainer Class

Inheritance: public ACContainer

```
#include <acapi/acrule.hh>
```

Data Members: No public data members are declared in this class.

The ACRuleContainer class is a container for all the rules defined in access control. This class does not directly represent any GDMO object class.

5.23.1 Constructor

```
ACRuleContainer()
```

The constructor creates a rule container object, so that when it is instantiated it contains all the rules that are subject to access control. If such an object has already been created, the internal reference count for the object is incremented by one.

5.23.2 Destructor

```
~ACRuleContainer()
```

5.23.3 ACRuleContainer Member Functions

`get_access_control_rules`

```
ACAccessControlRules get_access_control_rules()
```

Returns the `ACAccessControlRules` object that represents the `emAccessControlRules` GDMO object class. This `ACAccessControlRules` object can be used to get the various attributes of the `emAccessControlRule`.

`get_all_rules`

```
ACRuleList get_all_rules()
```

Returns all the rules available in the rule container object.

`get_group_rule_list`

```
ACRuleList get_group_rule_list(const RWCString& groupname)
```

Returns a list of all the rules that reference the given group.

`get_rule`

```
ACRule get_rule(const RWCString& rulename)
```

Returns the rule object whose name is specified by `rulename`.

ACScope Class

get_targets_rule_list

```
ACRuleList get_targets_rule_list(const RWCString& targetsname)
```

Returns all the rules that reference the given target.

5.24 ACScope Class

Inheritance: None

```
#include <acapi/actargets.hh>
```

Data Members: The following public data members are defined.

- type can be:
 - BASE_OBJECT
 - NTH_LEVEL
 - BASE_TO_NTH_LEVEL
 - ALL_LEVELS
 - ALL_LEVELS_EXCEPT_BASE
- level can be any positive integer.

The ACScope class contains the scope information.

5.24.1 Constructors

Default Constructor

```
ACScope( )
```

The default constructor initializes type to BASE_OBJECT and level to 0.

```
ACScope(MessScopeType t, U32 l)
```

The preceding constructor initializes type to *t* and level to *l*.

- type can be:
 - BASE_OBJECT
 - NTH_LEVEL
 - BASE_TO_NTH_LEVEL
 - ALL_LEVELS
 - ALL_LEVELS_EXCEPT_BASE

```
ACScope(const ACScope& other)
```

The preceding constructor is a copy constructor.

5.24.1.1 ACScope Operator Overloading

```
ACScope& operator=(const ACScope& other)
```

The preceding operator overloads the assignment operator and assigns the values of right side to left side.

5.24.1.2 ACScope Member Functions

No public member functions.

5.25 ACTargets Class

Inheritance: public ACOBJECT

```
#include <acapi/actargets.hh>
```

Data Members: No public data members are declared in this class.

The ACTargets class represents X.741's targets GDMO object class. Targets identify managed objects within the security domain.

5.25.1 Constructor

```
ACTargets(const RWCString& objectname, ACTargetsType type =
X741_TARGETS)
```

The constructor creates an `ACTargets` object with the given name and type. The default type is `X741_TARGETS`, which creates a `targets` GDMO object from X.741. Any other type creates an `emTargets` GDMO object which is defined in the Solstice EM Access Control module. If an object with the same name exists, the internal reference count for the object is incremented by one.

The constructor creates an `ACTargets` object with the given name and type. The type is either the `targets` GDMO object defined in `X741.gdmo` or the `emTargets` GDMO object defined in the Solstice EM Access Control module. The default type is the `targets` GDMO object if no type is specified.

5.25.2 Destructor

```
virtual ~ACTargets()
```

5.25.3 ACTargets Member Functions

`add_moc`

```
Result add_moc(const RWCString& mocname)
```

Adds the given managed object class (MOC) name to the MOC list that is defined for the target.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

If the given MOC already exists, the function sets the error type to `ACC_MOC_EXISTS` and the error string to “MOC exists in targetsList!”.

add_moi

```
Result add_moi(const RWCString& moiname)
```

Adds the given managed object instance (MOI) to the list of MOIs that are defined for the target.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

If the given MOI already exists in the target list, this function sets the error type to `ACC_MOI_EXISTS` and the error string to “MOI exists in targetsList!”.

destroy

```
Result destroy()
```

Removes the target from all the rules that reference it, and deletes the target.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

get_filter

```
ACFilter get_filter()
```

Returns the stored filter for the target.

get_moc_list

```
ACMOCList get_moc_list()
```

Returns the list of MOCs that are defined for the instantiated `target` object.

ACTargets Class

get_moi_list

```
ACMOIList get_moi_list()
```

Returns a list of the MOIs that are defined for the target.

Check for the error type to get any error that occurred while performing this function.

get_operations_list

```
ACOperationsList get_operations_list()
```

Returns the list of operations that are defined for this target. For example, {action, get, multipleObjectSelection, filter}.

Check for the error type to get any error that occurred while performing this function.

get_scope

```
ACScope get_scope()
```

Returns the target's scope.

remove_moc

```
Result remove_moc(const RWCString& mocname)
```

Removes the given MOC from this target's list of MOCs.

Returns TRUE on successful completion; otherwise, FALSE.

If the given MOC already exists, this function sets the error type to ACC_MOC_NOT_EXISTS and the error string to "MOC doesn't exist in targetsList!".

remove_moi

```
Result remove_moi(const RWCString& moiname)
```

Removes the given MOI from this target's list of MOIs.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

If the given MOI already exists, this function sets the error type to `ACC_MOI_NOT_EXISTS` and the error string to "MOI doesn't exist in targetsList!".

set_filter

```
Result set_filter(const ACFilter& filter)
```

Sets the given filter for the target object.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

set_moc_list

```
Result set_moc_list(ACMOCList& moclist)
```

Sets the given MOC list for the target object.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

set_moi_list

```
Result set_moi_list(ACMOIList& moilist)
```

Sets the given MOI list for the target object.

Returns `TRUE` on successful completion; otherwise, `FALSE`.

ACTargetsContainer Class

set_operations_list

```
Result set_operations_list(ACOperationsList& operlist)
```

Sets the given operations list for the target object.

Returns TRUE on successful completion; otherwise, FALSE.

set_scope

```
Result set_scope(const ACScope& scope)
```

Sets the given scope for the target object.

Returns TRUE on successful completion; otherwise, FALSE.

5.26 ACTargetsContainer Class

Inheritance: public ACContainer

```
#include <acapi/actargets.hh>
```

Data Members: No public data members are declared in this class.

The ACTargetsContainer class represents the targetContainer GDMO object class which is defined in the Solstice EM Access Control module.

ACTargetsContainer is a container for all the Solstice EM access control targets.

5.26.1 Constructor

```
ACTargetsContainer()
```

The constructor creates a targets container object to contain all the targets that are subject to access control. If such an object exists, the internal reference count for the object is incremented by one.

5.26.2 Destructor

```
~ACTargetsContainer()
```

5.26.3 ACTargetsContainer Member Functions

`get_all_targets`

```
ACTargetsList get_all_targets()
```

Returns all the targets under the `targetContainer` object.

`get_em_targets`

```
ACEMTargets get_em_targets(const RWCString& targetsname)
```

Returns the `ACEMTargets` object with the given name.

`get_targets`

```
ACTargets get_targets(const RWCString& targetsname)
```

Returns the `ACTargets` object with the given name.

5.27 ACUser Class

Inheritance: None

```
#include <acapi/acaccessuserlist.hh>
```

Data Members: No public data members are declared in this class.

The ACUser class stores a user's login name and full name in memory. Access control applications get a user's login name and full name by accessing the instantiated ACUser object.

5.27.1 Constructors

Default Constructor

```
ACUser()
```

The default constructor does not initialize the login name and full name.

```
ACUser(const RWCString& loginname, const RWCString& fullname)
```

The preceding constructor initializes the login name to *loginname* and the full name to *fullname*.

```
ACUser(const RWCString& loginname)
```

The preceding constructor initializes the login name to *loginname* and the full name to an empty string.

```
ACUser(const ACUser& other)
```

The above constructor is a copy constructor.

5.27.2 ACUser Operator Overloading

```
ACContainerData& operator = (const ACContainerData& other)
```

The preceding operator overloads the assignment operator, and assigns the values of *other* to *self*.

```
friend Boolean operator == (const ACContainerData& self, const  
ACContainerData& other)
```

The preceding operator overloads the equality operator, so that if both the *self* and *other* objects have the same login name, the overload method returns TRUE; otherwise, the method returns FALSE.

5.27.3 ACUser Member Functions

get_full_name

```
RWCString get_full_name() const
```

Returns the full name stored in the object.

get_login_name

```
RWCString get_login_name() const
```

Returns the login name stored in the object.

ACUser Class

is_valid_user

```
Boolean is_valid_user(unsigned int& error_code)
```

Verifies the validity of the instantiated ACUser by contacting em_login daemon to check whether the user is a valid user on the MIS host. Returns TRUE on successful completion; otherwise, FALSE.

set_full_name

```
void set_full_name(const RWCString&)
```

Sets the full name in the object to the given name.

set_login_name

```
void set_login_name(const RWCString&)
```

Sets the login name in the object to the given name.

Access Control Engine API

The Access Control Engine (ACE) API functions enable you to invoke Access Control Decision Function (ADF) and Access Control Enforcement Function (AEF) operations. The ADF and AEF are implemented in the ACE module, which is in turn implemented as a shared library.

The ACE API is designed using the low-level PMI and is independent of the MIS architecture. This allows the API to be plugged into different Auxiliary Servers (such as MIS and MPA), making it possible for the Auxiliary Servers to impose access control on the objects they manage.

This chapter comprises the following topics:

- Section 6.1 “Symbolic Constants” on page 6-2
- Section 6.2 “ACE API Classes” on page 6-3
- Section 6.3 “ACE Class” on page 6-3

6.1 Symbolic Constants

6.1.1 ACEOperationType

```
enum ACEOperationType {
    ACEAction,
    ACECreate,
    ACEDelete,
    ACEGet,
    ACESet,
    ACEReplace = 4,
    ACEAddMember = 4,
    ACERemoveMember = 4,
    ACEReplaceWithDefault = 4,
    ACEMultipleObjectSelection = 8,
    ACEFilter,
    ACEEventReport,
    ACEMaxNumOperations
}
```

The `ACEMaxNumOperations` element is a count of `ACEOperationTypes` and *not* an `ACEOperationType`.

6.1.2 ACEEnforcementAction

```
enum ACEEnforcementAction {
    DenyWithResponse,
    DenyWithoutResponse,
    AbortAssociation,
    DenyWithFalseResponse,
    Allow,
    Unknown,
    MaxNumEnforcementActions
}
```

The `MaxNumEnforcementActions` element is a count of the `ACEEnforcementActions` and *not* an `ACEEnforcementAction`.

6.2 ACE API Classes

This section describes the following ACE API classes:

- ACE
- ACEContext
- ACEDecision
- ACEDomain
- ACEGlobals
- ACEReqData
- AuxServerUtils

6.3 ACE Class

Inheritance: None

```
#include <ace/ace.hh>
```

Data members: No public data members are declared in this class.

This class abstracts the concept of ACE initialization and acts as a single point of access to the services provided by the ACE library. There can only be one object instance of this class for each security domain. This restriction helps in the management of and clear separation between an auxiliary server (such as MIS or MPA) and the Access Control Servers (ACS) to which it connects.

6.3.1 Constructor

```
ACE(const ACEDomain& domain, AuxServerUtils& aux_server_utils)
```

This constructor initializes the ACE library. It sets the security domain for which access control is to be provided to *domain*. It also takes a reference to the class derived from *AuxServerUtils*, which needs to be implemented as part of using ACE API.

Note – You should either create the ACE object on the heap or make sure it does not go out of scope.

6.3.2 Destructor

```
~ACE( )
```

6.3.3 ACE Member Functions

`get_ace_instance`

```
ACE& get_ace_instance()
```

This function is a static method. It returns the currently instantiated ACE object instance.

`check_access`

```
ACEDecision check_access(  
    const ACEReqData& ace_req_data) const throw(ACEException)
```

```
ACEDecision check_access(  
    const Message* msg,  
    const void* user_data = 0,  
    constRWTValSlist<Asn1Value>&in_oc_component_list=  
        ACEGlobals::null_oc_component_list,  
    const ACEContext& context =ACEGlobals::null_ace_context,  
    Boolean x740_supported = TRUE  
) const throw(ACEException)
```

Checks access for the user found in the *msg*'s access field and returns the ACEDecision object from which the enforcement action can be obtained. TABLE 6-1 describes this function's parameters.

TABLE 6-1 check_access() Parameters

Parameter	Description
<i>ace_req_data</i>	Reference to a ACEReqData object.
<i>msg</i>	Message for which access control needs to be applied.
<i>user_data</i>	User data to be passed to AuxServerUtils methods. This is analogous to the user_data parameter passed for typical callbacks.
<i>in_oc_component_list</i>	<p>List of all possible object classes starting from the root of the above oi. This list contains the following:</p> <ul style="list-style-type: none">• Object class root for /• Object class system for /systemId=name:"netareno• Object class log for /systemId=name:"netareno"/logId=string:"Alarmlog" <p>You can pass the default value ACEGlobals::null_oc_component_list ao, while applying the access control decisions, the ACE API calls AuxServerUtils::determine_class() to get the object class information for the message.</p>
<i>context</i>	A reference to an ACEContext object.

Note – The Message class is the base class used for messages passed between SAPs and the PMI.

`hi_process_ace_event`

```
void hi_process_ace_event(Ptr user_data, Ptr call_data)
```

The ACE API callback that processes all access control-related events once the application/auxiliary server is up and running. This method can be registered using the `Platform::when()` method. While registering with `Platform::when()`, the pointer to the ACE object should be passed in the `userdata` of the `Platform::when()` method, so that the `hi_process_ace_event()` receives a pointer to the ACE object in its `user_data` parameter.

This is a static method that processes high-level events. `call_data` must be a `CurrentEvent` object pointer. For more information about the `CurrentEvent` class.

`lo_process_ace_event`

```
void lo_process_ace_event(Ptr user_data, Ptr call_data)
```

The ACE API callback that processes all access control-related events once the application or auxiliary server is up and running. This method is typically called when there is an existing callback on the `Platform`. When the `Platform` callback is called, you can call this method with `user_data` containing a pointer to the ACE object and `call_data` containing the message from `CurrentEvent`.

This method is a static method that processes low-level events. `call_data` must be a `Message` object pointer.

6.4 ACEContext Class

```
#include <ace/ace_context.hh>
```

The `ACEContext` class stores the context of an access check on a scoped and filtered CMIP request that cannot be resolved deterministically.

6.4.1 Constructor

```
ACEContext(const Message* p_orig_user_req = 0)
```

This constructor creates an `ACEContext` object with the original user request set to `p_orig_user_req`. If `p_orig_user_req` is not provided, the original user request is set to 0.

6.4.2 Destructor

```
~ACEContext()
```

6.4.3 ACEContext Operator Overloading

```
operator void*() const
```

```
int operator !() const
```

6.4.4 ACEContext Member Functions

```
get_orig_user_req
```

```
Message* get_orig_user_req() const
```

Returns the original user request message that the instantiated `ACEContext` object contains.

ACEContext Class

get_scope

```
MessScope& get_scope() const
```

Returns the scope that the instantiated ACEContext object contains.

get_filter

```
Asn1Value& get_filter() const
```

Returns the filter information that the instantiated ACEContext object contains.

set_scope

```
void set_scope(const MessScope& scope)
```

Sets the scope of the instantiated ACEContext object to *scope*.

set_filter

```
void set_filter(const Asn1Value& filter)
```

Set the filter of the instantiated ACEContext object to *filter*.

6.5 ACEDecision Class

```
#include <ace/ace_decision.hh>
```

The `ACEDecision` class abstracts the concept of ADF decision in response to an access check request to ACE.

6.5.1 Constructor

```
ACEDecision()
```

The preceding constructor creates an `ACEDecision` object.

```
ACEDecision(const ACEDecision& rhs)
```

The preceding constructor is a copy constructor.

```
ACEDecision(
    const AsnlValue& oc,
    const AsnlValue& oi,
    const ACEEnforcementAction& enforcement_action,
    const Message* p_msg = 0
)
```

This constructor creates an `ACEDecision` object and initializes its “proposed response” data member as follows:

- If *enforcement_action* is `DenyWithResponse`, the proposed response is set to `ACCESS_DENIED`.
- If *enforcement_action* is `DenyWithoutResponse` or `AbortAssociation`, the proposed response is set to `PROCESS_FAILURE`.

6.5.2 Destructor

```
~ACEDecision()
```

6.5.3 ACEDecision Member Functions

`get_enforcement_action`

```
ACEEnforcementAction get_enforcement_action() const
```

Returns the enforcement action for the ACE decision. Refer to Section 6.1 “Symbolic Constants” on page 6-2,” for return values.

`get_proposed_response`

```
ObjResMess* get_proposed_response() const
```

Returns the proposed response of the ACEDecision object. The return value is an ObjResMess pointer to an ACCESS_DENIED or PROCESS_FAILURE message.

`get_proposed_response` returns NULL if the decision was *not* denied.

`get_original_request`

```
Message* get_original_request() const
```

Returns the original message given to ACE by the ACE client for which access control is to be applied.

6.6 ACEDomain Class

```
#include <ace/ace_domain.hh>
```

The `ACEDomain` class abstracts the concept of a security domain and information that is needed to identify and access the domain server ACS.

6.6.1 Constructor

```
ACEDomain(const DataUnit& domain_name, MessageSAP* p_sap)
```

The preceding constructor creates an `ACEDomain` object with the domain name set to *domain_name* and the message SAP set to *p_sap*.

```
ACEDomain(const ACEDomain& that)
```

The preceding constructor is a copy constructor.

6.6.2 Destructor

```
~ACEDomain()
```

6.6.3 ACEDomain Member Function

`get_sap`

```
MessageSAP* get_sap() const
```

Returns the message SAP of the `ACEDomain` object.

6.7 ACEReqData Class

```
#include <ace/ace_req_data.hh>
```

The ACEReqData class stores operation request data.

6.7.1 Constructor

```
ACEReqData(
    const Asn1Value& user_id =ACEGlobals::null_asn1_value,
    const Asn1Value& oc = ACEGlobals::null_asn1_value,
    const Asn1Value& oi = ACEGlobals::null_asn1_value,
    const ACEOperationType& operation =

    ACEGlobals::null_operation_type,
    const MessScope& scope = ACEGlobals::null_scope,
    const Asn1Value& filter = ACEGlobals::null_asn1_value,
    const void* user_data = 0,
    const RWTValSlist<Asn1Value>& in_oc_component_list =

    ACEGlobals::null_oc_component_list,
    const Message* opt_original_msg = 0,
    ACEContext context = ACEGlobals::null_ace_context,
    Boolean x740_supported = TRUE
)
```

TABLE 6-2 describes the parameters of the preceding constructor.

TABLE 6-2 ACEReqData() Constructor Parameters

Parameter	Description
<i>user_id</i>	An encoded GraphicString whose access needs to be checked against a request.
<i>oc</i>	Object class of the message.
<i>oi</i>	Object instance of the message.
<i>operation</i>	Type of the message. Values are obtained from ACEOperationType.

TABLE 6-2 `ACEReqData()` Constructor Parameters (*Continued*)

Parameter	Description
<i>scope</i>	Scope of the message.
<i>filter</i>	Filter of the message.
<i>user_data</i>	User data to be passed to <code>AuxServerUtils</code> methods. This is analogous to the <code>user_data</code> parameter passed for typical callbacks.
<i>in_oc_component_list</i>	<p>List of all possible object classes for the message that is being passed to <code>check_access</code>.</p> <p>For example, if the message being passed is a get request whose <code>oi = /systemId=name:"netareno"/</code> <code>logId=string:"Alarmlog",</code> <i>in_oc_component_list</i> contains all possible object classes starting from the root of the above <code>oi</code>. The list contains the following:</p> <ul style="list-style-type: none"> • Object class root for <code>/</code> • Object class system for <code>/systemId=name:"netareno</code> • Object class log for <code>/systemId=name:"netareno"/</code> <code>logId=string:"Alarmlog"</code> <p>The default value for this parameter is <code>ACEGlobals::null_oc_component_list</code>. If the default value is used, ACE API calls the <code>AuxServerUtils::determine_class()</code> to get the object class information for the message.</p>
<i>opt_original_msg</i>	Message for which access control needs to be applied.
<i>context</i>	A reference to an <code>ACEContext</code> object.

6.7.2 Destructor

```
~ACEReqData();
```

6.8 AuxServerUtils Class

Inheritance: None

```
#include <ace/ace_aux_server_utils.hh>
```

Data members: No public data members are declared in this class.

The C++ class, `AuxServerUtils`, is an abstract base class that provides auxiliary services to ACE objects.

6.8.1 Constructor

```
AuxServerUtils()
```

The `AuxServerUtils` class is an abstract base class, and cannot be instantiated.

6.8.2 Destructor

```
~AuxServerUtils()
```


6.8.3 AuxServerUtils Virtual Functions

check_filter

```
FilterResult check_filter(  
    const Asn1Value& oi,  
    const Asn1Value& filter,  
    const void* userdata = 0)
```

This method is a hook into the application or auxiliary server's code called by the ACE. This method is called whenever the ACE is evaluating a target that has a filter specified in it. This method is expected to check whether *oi* satisfies the *filter*. *userdata* has a valid pointer if ACE::check_access() was called with *userdata* in it.

The possible return values are NO_MATCH, MATCH, INVALID, RESOURCELIM, and PROCESSFAIL as defined in pmi/filter.hh. If the filter field of the targets is not being used, this method returns MATCH.

determine_class

```
Asn1Value determine_class(  
    const Asn1Value& oi,  
    const void* userdata = 0)
```

This method is a hook into the application or auxiliary server's code called by the ACE. This method is called whenever the ACE does not find the managed object class of the request in the target's object classes list. This method evaluates the object class of the given *oi* and returns it. *userdata* has a valid pointer if ACE::check_access() was called with *userdata* in it.

`aux_get_req`

```

Result  aux_get_req(
    MessageSAP* sap,
    const Asn1Value& oi,
    const MessScope& scope,
    const Callback& cb,
    const Asn1Value& oc = ACEGlobals::actual_class_oc
    const Asn1Value&
        attrs = ACEGlobals::null_asn1_value,
    const Asn1Value&
        filter = ACEGlobals::null_asn1_value,
    const Asn1Value&
        access = ACEGlobals::null_asn1_value,
    const MessSync& sync = BEST_EFFORT,
    MessId& op_id = ACEGlobals::null_msg_id,
    const U32 flags = 0,
    const Boolean& sub_trans = FALSE)

```

This method is called by ACE internals to send a GET_REQ to the MIS to load access control information inside the ACE.

`extract_message`

```

Message*  extract_message(Ptr call_data)

```

This method is called by ACE internals to receive the response to a GET_REQ sent in by `aux_get_req`.

`aux_check_create_filter`

```

FilterResult aux_check_create_filter(
    const Message* msg,
    const Asn1Value& filter)

```

This method is a hook into the application or auxiliary server's code called by the ACE. This method is called whenever the ACE is evaluating a create request against a target that has a filter specified in it. This method checks whether *msg* satisfies the *filter*. The possible return values are NO_MATCH, MATCH, INVALID, RESOURCELIM, and PROCESSFAIL as defined in `pmi/filter.hh`.

`aux_check_event_filter`

```
FilterResult aux_check_event_filter(  
    const Message* msg,  
    const Asn1Value& filter)
```

This method is a hook into the application or auxiliary server's code called by the ACE. This method is called whenever the ACE is evaluating an event request. This method checks whether the event *msg* satisfies *filter*. The possible return values are NO_MATCH, MATCH, INVALID, RESOURCELIM, and PROCESSFAIL as defined in `pmi/filter.hh`.

Nerve Center Interface

The Nerve Center Interface (NCI) library is built on top of the Portable Management Interface (PMI). The NCI library allows applications to create template requests, launch the request against Management Information Server (MIS) objects, and retrieve information about objects.

This chapter comprises the following topics:

- Section 7.1 “Requests” on page 7-1
- Section 7.2 “Class and Function Summary” on page 7-2
- Section 7.3 “NC Requests” on page 7-3
- Section 7.4, “NCI Library Classes
- Section 7.8 “NCI Library Functions” on page 7-11
- Section 7.10.31.2 “Event Request Example” on page 7-30

7.1 Requests

A Nerve Center (NC) template is a type of management request; it is used to manage a set of selected objects in the MIS. A NC template is comprised of state, condition, transition, and action requests. NC template definitions reside in the MIS.

When a request is launched, the Nerve Center polls remote objects and retrieves the requested information. If the application transitions to a new state, the managed object’s severity and state attributes are changed in the MIS and appropriate applications are informed of that object’s state change.

Note – When compiling NCI clients, do not disable C++ exceptions (`-noex` option with Sun C++ compilers) and do link NCI clients with the RogueWave library.

7.2 Class and Function Summary

TABLE 7-1 summarizes the classes and functions included this chapter.

TABLE 7-1 Nerve Center Classes and Functions

Class/Function	Return Value	Description
TABLE 7-2 lists the library classes included in the NCI.		Allows iteration through the responses received for an asynchronous launch
NCParsedReqHandle Class		Parses request handles returned by NCI functions
NCTopoInfoList Class		Builds a list of toponode information
nci_action_add	Result	Adds the action to the transition
nci_action_delete	Result	Deletes the action from the list of actions for transition
nci_async_request_start	Waiter	Launches NC requests asynchronously
nci_condition_add	Result	Creates a condition object that can be used for building NC templates
nci_condition_delete	Result	Deletes an existing condition
nci_condition_get	char*	Returns an ASCII string containing RCL statement(s)
nci_init	Result	Initializes the NCI library
nci_parse_handle	Result	Parses a given request handle and returns data
nci_pollrate_add	Result	Creates a new pollrate object in the MIS
nci_pollrate_delete	Result	Deletes an existing pollrate object in the MIS
nci_request_delete	Result	Deletes a running request
nci_request_dump	Array(DU)	Returns current state, severity, and variable data
nci_request_info	Result	Returns state name, severity string, and severity value
nci_request_list	Array(DU)	Returns an array of request handles
nci_request_start	(DU)	Starts a management request against an object

TABLE 7-1 Nerve Center Classes and Functions (*Continued*)

Class/Function	Return Value	Description
nci_severity_add	Result	Creates a new security level in the MIS
nci_severity_delete	Result	Deletes an existing severity in the MIS
nci_state_add	Result	Adds the state to a NC template
nci_state_delete	Result	Deletes the state from a NC template
nci_state_get	NC_State	Gets the handle to identify the state in a NC template
nci_template_add	Result	Edits a NC template
nci_template_copy	Result	Copies a NC template
nci_template_create	Result	Creates a handle for a NC template
nci_template_delete	Result	Deletes an existing NC template from the MIS
nci_template_find	NC_Defn	Gets the handle for an existing request for editing
nci_template_revert	NC_Defn	Undoes changes done to the NC template
nci_template_store	Result	Stores the NC template
nci_transition_add	Result	Adds a transition to an existing NC template
nci_transition_delete	Result	Deletes a transition between two states
nci_transition_find	NC_Transition	Gets the transition in a NC template
nci_transition_get	NC_Transition	Gets the handle on transition from the state

7.3 NC Requests

7.3.1 Synchronous Launches

Synchronous launches imply that associated functions do not return until after all the parameter validation and communication with the MIS, regarding the start of the request, is complete. These functions do not actually confirm whether the request has started successfully or not. In that sense, it is asynchronous.

NC Requests

In comparison, functions in an asynchronous launch return even before any communication with the MIS, regarding the start of the request, takes place.

The functions that are specifically synchronous are:

- `nci_request_delete`
- `nci_request_dump`
- `nci_request_info`
- `nci_request_list`
- `nci_request_start`

Many of the request functions use the variable, *handle*. *handle* is an optional user-defined string. When it is used, the NCI uses it to build the request handle that the NCI returns.

The purpose of using *handle* is to allow an application, like the auto daemon, to get a list of running requests from the NCI and know which ones it has started. Such a list is useful when an application restarts. An application does not *have* to use *handle*; it can keep a list of the requests it has started instead. If *handle* is not passed, the request handle is formed by the NCI without it

7.3.2 Asynchronous Launches

In asynchronous launches, associated functions return before communication with the MIS, regarding the start of the request, takes place. In comparison, synchronous launches imply that associated functions do not return until after all the parameter validation and communication with the MIS, regarding the start of the request, is complete.

The classes/functions associated with asynchronous launches are:

- TABLE 7-2 lists the library classes included in the NCI.
- `nci_async_request_start`
- `NCTopoInfoList` Class

7.4 NCI Library Classes

TABLE 7-2 lists the library classes included in the NCI.

TABLE 7-2 NCI Library Classes

NCI Classes

TABLE 7-2 lists the library classes included in the NCI.

NCParsedReqHandle Class

NCTopoInfoList Class

7.5 NCAsyncResIterator Class

The `NCAsyncResIterator` class allows you to iterate through the responses received for an asynchronous launch of possible multiple requests. Each iteration allows you to extract response information about one request.

7.5.1 Constructor

```
NCAsyncResIterator(Ptr call_data );
```

This constructor takes *call_data* as an argument. *call_data* is the data supplied by `nci_async_request_start()`, when `nci_async_request_start()` calls the user-installed callback.

7.5.2 Destructor

```
~NCAsyncResIterator();
```

7.5.3 Operator Overloading for Prefix Operator++

```
RWBoolean operator++();
```

The prefix increment operator advances the iterator one position in the response list. Returns `FALSE`, if it advances past the end of the response list. Otherwise, it returns `TRUE`.

Note – The postfix operator `operator++(int)` is not supported.

7.5.4 Member Functions

`get_req_handle`

```
const DataUnit& get_req_handle() const;
```

Returns the request handle for the request determined by the current position of the iterator in the response list. The results are undefined if the iterator is no longer valid or if `operator++()` has previously returned `FALSE`.

`get_req_status`

```
NCAsyncReqStatus get_req_status() const;
```

Returns the status of the request determined by the current position of the iterator in the response list. The results are undefined if the iterator is no longer valid or if `operator++()` has previously returned `FALSE`.

`NCAsyncReqStatus` represents the status of the request that is launched asynchronously.

```
enum NCAsyncReqStatus
{
    NOT_INITIALIZED,
    AWAITING_RESPONSE,
    LAUNCH_SUCCESS,
    LAUNCH_FAILURE
};
```

`NOT_INITIALIZED`: Indicates the NC request was never launched.

`AWAITING_RESPONSE`: Indicates that a request (internal CMIP request) has been made to the MIS to launch the NC request, and is waiting for a response from the MIS.

`LAUNCH_SUCCESS`: The NC request was successfully launched.

`LAUNCH_FAILURE`: The NC request launch failed.

`get_error_reason`

```
const RWCString& get_error_reason() const;
```

Returns the reason for failure to launch a given request, determined by the current position of the iterator in the response list. You should use this member function only when `get_req_status()` returns `LAUNCH_FAILURE`.

The results are undefined if the iterator is no longer valid or if `operator++()` has previously returned `FALSE`.

7.6 NCParsedReqHandle Class

`NCParsedReqHandle` class is responsible for parsing request handles returned by NCI functions. For backward compatibility, `nci_parse_handle()` is retained. However, it is recommended that `NCParsedReqHandle` be used wherever possible. This provides better insulation against any changes to the request handle implementation and more information can be extracted from the request handle.

7.6.1 Constructors

```
NCParsedReqHandle(const DataUnit& req_handle_du);
```

This constructor parses the *req_handle_du* which is received by NCI functions that launch requests, either synchronously or asynchronously.

If the request handle passed is invalid, the `NCParsedReqHandle` constructor throws an `NCEException` to the user. To determine the cause of the exception, use `NCEException::why()` which returns `char*`.

7.6.2 Default Destructor

```
~NCParsedReqHandle();
```

7.6.3 Member Functions

`get_topo_id`

```
u_long get_topo_id() const;
```

This function returns the `topo_id` from the `request_handle`.

`get_mis_name`

```
const RWCString& get_mis_name() const;
```

This function returns the name of the MIS against which the request was launched.

get_template_name

```
const RWCString& get_template_name() const;
```

This function returns the name of the NC template that was used to launch the request.

get_invoke_id

```
u_long get_invoke_id() const;
```

This function returns the invoked id of the asynchronous launch. Invoke ID is per process and per invocation of `nci_async_request_launch()`. For synchronous launches, `invoke_id` is always zero.

get_user_stub

```
const RWCString& get_user_stub() const;
```

This function returns the *user_stub* that was used in the construction of the request handle. This *user_stub* is passed to the invocation of synchronous versions of request launches.

7.7 NCTopoInfoList Class

NCTopoInfoList class is used to build a list of toponode information. Information about individual toponodes is shared using the copy constructor and assignment operator.

7.7.1 Default Constructor

```
NCTopoInfoList();
```

This function is the default constructor that constructs an empty topology information list.

7.7.2 Copy Constructor

```
NCTopoInfoList (const NCTopoInfoList& that);
```

This is the copy constructor that increments the reference count on the internal implementation object contained within *that* object.

7.7.3 Destructor

```
~NCTopoInfoList();
```

This destructor decrements the reference count on the internal implementation object and deletes the internal implementation object when the reference count goes to zero.

7.7.4 Operator Overloading for Operator=

```
NCTopoInfoList& operator=(const NCTopoInfoList& that);
```

The assignment operator increments the reference count on the internal implementation object contained within *that* object.

7.7.5 Member Functions

`add_topo_info`

```
Result add_topo_info(
    u_long topo_id,
    Array(DU)& fdn_set,
    CDU mis_name = null_du
);
```

Alternately,

```
Result add_topo_info(
    u_long topo_id,
    CDU mis_name = null_du
);
```

This member function adds information about one toponode to the `NCTopoInfoList` object. This overloaded member function is provided for the sake of efficiency and performance. Some applications might already cache the fdns of the toponodes (the value of the `topoNodeMOSet` GDMO attribute).

Such applications can pass in the *fdn_set* (optional). Each fdn in the *fdn_set* should be either in the slash or brace format. Because `nci_async_request_start()` does not issue any requests to the MIS to get information about the toponodes, performance gains are achieved.

If *mis_name* is not specified, it is assumed to be the local MIS (the MIS to which `nci_init()` initially connected). Returns `OK` if successful. Otherwise, it returns `NOT_OK`.

7.8 NCI Library Functions

Note – All functions that return `Result` are either `OK` or `NOT_OK`. All functions that return `Boolean` are either `TRUE` or `FALSE`.

Several NCI functions use an argument *mis_name*, as the NCI supports MIS to MIS awareness. NCI accepts *mis_name* in the following formats:

NCI Library Functions

- slash format (/systemId=name:"sol")
- string name ("sol")

The *mis_name* is the name of the MIS on which the given NCI function is effective. Unless otherwise specified, *mis_name* automatically maps to the local MIS that the NCI application is connected to. *null_du* is the default value, and represents the empty DataUnit.

TABLE 7-3 lists the NCI library functions.

TABLE 7-3 Nerve Center Library Functions

Function	Function
nci_action_add	nci_severity_delete
nci_action_delete	nci_state_add
nci_async_request_start	nci_state_delete
nci_condition_add	nci_state_get
nci_condition_delete	nci_template_add
nci_condition_get	nci_template_copy
nci_init	nci_template_create
nci_parse_handle	nci_template_delete
nci_pollrate_add	nci_template_find
nci_pollrate_delete	nci_template_revert
nci_request_delete	nci_template_store
nci_request_dump	nci_transition_add
nci_request_info	nci_transition_delete
nci_request_list	nci_transition_find
nci_request_start	nci_transition_get
nci_severity_add	

7.9 NCI Global Variables

7.9.1 `nci_error_reason`

```
extern DataUnit nci_error_reason;
```

NCI has a facility that allows you to diagnose errors using `nci_error_reason`. This global variable, of the `DataUnit` type, can be used to know the reason for any failures of any NCI function. This variable contains a meaningful reason only immediately after any NCI function returns an error. This variable is not re-initialized by NCI every time an NCI function is invoked. It is set only when NCI encounters an error.

7.9.2 `topoNodeId` Argument

Several of the NCI functions include the argument *topoNodeId*. NCI accepts *topoNodeId* in the form of a `DataUnit` and supports either of the following formats:

```
u_long topoNodeId;
```

```
DataUnit duTopoNodeId = DataUnit::printf("%u", topoNodeId);
```

The `topoNodeId` should be constructed as shown in the above syntax before passing to NCI any of the functions that require *topoNodeId*. You can use `sscanf()` or `sprintf()` to simulate a comparable effect.

- *mis_name:topoId*
- *topoId*

7.10 NCI Functions

7.10.1 `nci_action_add`

```
Result
nci_action_add(
    NC_Transition &transq,
    const char *action,
    const char *arg0,
    const char *arg1,
    CDU mis_name= null_du
);
```

The function `nci_action_add()` adds the action *action* to the transition *transq*. The action can be either a condition, or mail, or unixcmd. For condition, *arg0* is the name of the condition and *arg1* is null. For mail, *arg0* is the address and *arg1* is the message. For unixcmd, *arg0* is the UNIX command name and *arg1* is the argument. *transq* is the handle on transition as returned by the `nci_transition_find()` function. The function returns `TRUE` if the *action* is added, `FALSE` if there is an error.

7.10.2 `nci_action_delete`

```
Result
nci_action_delete(
    NC_Transition &transq,
    const char *action,
    const char *arg0,
    const char *arg1
);
```

The function `nci_action_delete()` deletes the action *action* from the list of actions for transition *transq*. The action can be either a condition, or mail, or unixcmd. For condition, *arg0* is the name of the condition and *arg1* is null. For mail, *arg0* is the address and *arg1* is the message. For unixcmd, *arg1* is the list of arguments specified by *arg0*. *arg1* is in string format. *transq* is the handle on transition as returned by the function `nci_transition_find()`. The function returns `TRUE` if the action is deleted, `FALSE` if there is an error.

7.10.3 nci_async_request_start

```

Waiter
nci_async_request_start(
    const char*      template_name,
    NCTopoInfoList   topo_info_list,
    Callback          user_cb,
    Timeout           timeout_time = 3600.0 seconds
);

```

`nci_async_request_start()` returns a waiter that represents the asynchronous launch.

template_name is the name of the NC template that is used to launch the request.

topo_info_list is information about the toponodes against each of which the request is to be launched asynchronously.

user_cb is the callback, if installed by the user, that is called when the asynchronous launch completes, either successfully or not. *user_cb* is called by NCI. The *call_data* parameter of the callback is a pointer that must be used to construct `NCAsyncResIterator` object. If the *call_data* pointer is not used to construct the `NCAsyncResIterator` object, a memory leak may occur.

timeout_time is the timeout interval after which `nci_async_request_start()` times out and returns. Currently, it is recommended that you use the default timeout. Timeout for lower timer intervals is not supported in this release.

Note – Waiter cancellation or waiter clobbering is not supported.

7.10.4 nci_condition_add

```
Result
nci_condition_add(
    const char *condition_name,
    const char *condition_desc,
    const char *condition,
    CDU mis_name = null_du
);
```

The function `nci_condition_add()` creates a condition object that can then be used for building NC templates. The condition itself is one or more lines separated by new lines specified in the Request Condition Language (RCL).

7.10.5 nci_condition_delete

```
Result
nci_condition_delete(
    const char *condition_name,
    CDU mis_name = null_du
);
```

The function `nci_condition_delete()` deletes an existing condition. If a condition is associated with a transition it cannot be deleted until the transitions are deleted.

7.10.6 nci_condition_get

```
char*
nci_condition_get(
    const char *condition_name,
    char *conditionDesc,
    CDU mis_name = null_du
);
```

The function `nci_condition_get()` returns an ASCII string containing a Request Condition Language statement or statements.

7.10.7 nci_init

```
Result
nci_init(
    const char *location,
    DU& errorMsg
);
```

```
Result
nci_init
    Platform &platform,
    DU& errorMsg
);
```

The function `nci_init()` is an NCI library initialization routine and takes either a hostname (*location*) or a platform object (*platform*).

The function that takes *location* as the argument connects to the MIS on the host specified by *location* and keeps the platform object internal. Of course, one could retrieve such a platform object using the static method `Platform::default_platform()`. The function returns `NOT_OK` if it could not connect to the MIS on the host specified by *location*. Otherwise, it returns `OK`.

The function that takes *platform* as the argument assumes the platform object is connected to some MIS and uses that platform object to talk to that MIS. If the platform object is valid and if the platform object is connected to a valid MIS, then the function returns `OK`. Otherwise, it returns `NOT_OK`.

7.10.8 nci_parse_handle

```
Result
nci_parse_handle(
    const DataUnit &request_handle,
    DataUnit &template_name,
    DataUnit &topoNodeId
);
```

Note – NCParsedReqHandle class is responsible for parsing request handles returned by NCI functions. For backward compatibility, `nci_parse_handle()` is retained. However, it is recommended that NCParsedReqHandle be used wherever possible. This provides better insulation against any changes to the request handle implementation and more information can be extracted from the request handle.

The function `nci_parse_handle()` parses a given request handle and returns the *template_name* and the information about *topoNodeId* against which the request was launched. The *request_handle* must be a valid one returned by a previous relevant NCI function. It returns OK if parsing succeeds. Otherwise, it returns NOT_OK and *template_name* and *topoNodeId* are undefined.

7.10.9 nci_pollrate_add

```
Result
nci_pollrate_add(
    const char *pollrate_name,
    int rate,
    CDU mis_name = null_du
);
```

The function `nci_pollrate_add()` creates a new pollrate object in the MIS. This pollrate can be used in creating NC templates. The pollrate is in seconds.

7.10.10 `nci_pollrate_delete`

```
Result
nci_pollrate_delete(
    const char *pollrate_name,
    CDU mis_name = null_du
);
```

The function `nci_pollrate_delete()` deletes an existing pollrate object in the MIS.

7.10.11 `nci_request_delete`

```
Result
nci_request_delete(
    CDU request_handle
);
```

The function `nci_request_delete()` deletes a running request. The handle for the request must be passed. The handle can be obtained from either `nci_request_start()` or by `nci_request_list()`. These request calls are unconfirmed and asynchronous.

7.10.12 `nci_request_dump`

```
Array(DU)
nci_request_dump(
    CDU request_handle
);
```

For the request identified by *request_handle*, the function `nci_request_dump()` returns the information about the current state, severity, and each variable's name, type, and value. For example, for three variables, eleven pieces of information are returned: current state, severity, and name, type, and value of each variable. The information is returned as the value of the function. If there is an error, an empty array is returned.

7.10.13 nci_request_info

```
Result  
nci_request_info(  
    CDU request_handle,  
    DU& statename,  
    DU& severity_name,  
    int *severity  
);
```

For the request identified by *request_handle*, the function `nci_request_info()` returns the name of state in *statename*, severity string in *severity_name*, and severity value in the integer pointed to by *severity*. The function returns `TRUE` if the request exists, `FALSE` otherwise.

7.10.14 nci_request_list

```
Array(DU)  
nci_request_list(  
    CDU mis_name = null_du  
);
```

The function `nci_request_list()` returns an array of request handles. `nci_parse_handle()` can be used to return the NC template name and `topoNodeId`.

7.10.15 nci_request_start

```
DU
nci_request_start(
    const char *template_name,
    CDU topoNodeId,
    const char *handle
);
```

The function `nci_request_start()` starts a management request with *template_name* against the managed object associated with *topoNodeId* in the topology part of the MIT. If the request starts successfully, the request handle is returned. This handle can be used, in the `nci_request_delete`, to stop and thereby delete the running request. If the request fails to start, a `null_du` is returned.

7.10.15.1 Alternative Syntax #1

```
Array(DU)
nci_request_start(
    const char *name,
    const Array(DU) &toponode_id_array,
    const char *handle
);
```

This function call performs the same job as the one above, except that multiple requests using the same NC template can be started against multiple `topoNodes`. One request is launched for each `topoNode` ID in the array. An array of handles is returned. Each handle in the array corresponds to a `topoNode` in the same index in the *toponode_id_array*.

7.10.15.2 Alternative Syntax #2

```
DU
nci_request_start(
    const char *template_name,
    CDU oi,
    CDU topoNodeId,
    const char *handle
);
```

This function call increases performance by specifying a Fully Distinguished Name with *oi*. The function does not have to query the MIS for the FDN.

7.10.15.3 Alternative Syntax #3

```
DU
nci_request_start(
    const char *template_name,
    const Array(DU) &oiSet,
    CDU topoNodeId,
    const char *handle
);
```

Performs the same job as the preceding function, except this function can take multiple FDNs and returns an array of handles.

7.10.15.4 Alternative Syntax #4

```
Array(DU)
nci_request_start(
    const char *template_name,
    const Array(DU) &oi_array,
    const Array (DU) &toponode_id_array,
    const char *handle
);
```

These function calls are analogous to the two preceding calls, except that they start requests against any managed object (or objects, for the second call), even if no TopoNode corresponds to that object. Returned values are as described for their TopoNode counterparts.

The DU is the ObjectInstance in either absolute-pathname or FDN-name format. The DU is the value of `$pollfdn` when the request is created. The second version, above, differs from the first in that it allows you to launch one request for each DU in the array.

An array of handles is returned. Each handle in the array corresponds to a `topoNode` in the same index in the `toponode_id_array`. If a given request failed to start, the handle in the returned array is a `null_du`.

7.10.15.5 Alternative Syntax #5

```
Array(DU)
nci_request_start(
    const char *template_name,
    const Array(Array(DU)) &oiSetArray,
    const Array(DU) &toponode_id_array,
    const char *handle
);
```

Performs the same job as the preceding function, except this function can take multiple FDNs.

7.10.16 nci_severity_add

```
Result
nci_severity_add(
    const char *severity_name,
    int severity,
    const char *color,
    CDU mis_name = null_du
);
```

The function `nci_severity_add()` creates a new severity level in the MIS. The severity level can be from 1 to 32000 and the color is any X11 color, such as red, purple, or yellow.

7.10.17 nci_severity_delete

```
Result
nci_severity_delete(
    char *severity_name,
    CDU mis_name = null_du
);
```

The function `nci_severity_delete()` deletes an existing severity in the MIS. The severity is deleted by name.

7.10.18 nci_state_add

```
Result
nci_state_add(
    NC_Defn &def,
    const char *state_name,
    const char *pollrate_name,
    const char *severity_name,
    const char *state_desc
);
```

The function `nci_state_add()` adds a state with name *state_name*, poll rate *pollrate_name*, severity *severity_name*, and description *state_desc* to the NC template identified by *def*. The function returns `TRUE` if the state is added, `FALSE` otherwise.

7.10.19 nci_state_delete

```
Result
nci_state_delete(
    NC_Defn &def,
    const char *state_name
);
```

The function `nci_state_delete()` deletes the state with name *state_name* from the NC template identified by *def*. The function returns `TRUE` if the state is deleted, `FALSE` otherwise.

7.10.20 nci_state_get

```
NC_State
nci_state_get(
    NC_Defn &def,
    const char *state_name
);
```

The function `nci_state_get()` gets the handle to identify the state with name *state_name* in the NC template identified by *def*. The function returns a valid `NC_State`, which is a handle on the Nerve Center state, and returns a default (invalid) `NC_State` if there is an error.

7.10.21 nci_template_add

```
Result
nci_template_add(
    NC_Defn &def,
    const char *name,
    const char *descr
    CDU mis_name = null_du
);
```

The function `nci_template_add()` is used when you have updated an existing NC template. The first time a NC template is created you use `nci_template_store()`. Subsequent edits require `nci_template_add()`. This function cannot be used to modify a NC template's name. An alternative is to use the function `nci_template_copy()` to copy the NC template using a different name, then delete the original NC template.

7.10.22 nci_template_copy

```
NC_Defn
nci_template_copy(
    NC_Defn &source_defn,
    const char *newname,
    const char *descr,
    CDU mis_name = null_du
);
```

The function `nci_template_copy()` returns a handle for a new copy of NC template name. The handle of the original NC template must be passed in.

7.10.23 nci_template_create

```
NC_Defn
nci_template_create(
    const char *template_name,
    const char *template_desc,
    CDU mis_name = null_du
);
```

The function `nci_template_create()` creates a handle for a NC template. After creation, the NC template can be built with other NCI library calls passing `NC_Defn` as the handle for the NC template.

7.10.24 nci_template_delete

```
Result
nci_template_delete(
    NC_Defn &def
);
```

The function `nci_template_delete()` deletes an existing NC template from the MIS. This call fails if any requests are currently using this NC template.

7.10.25 nci_template_find

```
NC_Defn
nci_template_find(
    const char *template_name,
    CDU mis_name = null_du
);
```

The function `nci_template_find()` is used to get the handle for an existing request for editing.

7.10.26 nci_template_revert

```
NC_Defn
nci_template_revert(
    const char *template_name,
    CDU mis_name = null_du
);
```

The `nci_template_revert()` function allows you to undo changes you have made to the NC template. This function is effective only if you have not invoked `nci_template_store()` on the NC template you are changing.

Typical usage is to get the NC template definition by invoking `nci_template_find()`, and make changes to the NC template definition (such as adding states, conditions, and transitions). If you need to undo any changes to the NC template, invoke `nci_template_revert()` before any call to `nci_template_store()` is made.

7.10.27 nci_template_store

```
Result
nci_template_store(
    NC_Defn &def,
    const char *template_name,
    const char *descr,
    CDU mis_name = null_du
);
```

The function `nci_template_store()` is called after the NC template has been completely built. If `nci_template_store()` returns `TRUE`, the NC template is ready for event management.

7.10.28 nci_transition_add

```
Result
nci_transition_add(
    NC_Defn &def,
    const char *from,
    const char *to,
    const char *condition,
    const char *action,
    const char *arg0,
    const char *arg1
);
```

The function `nci_transition_add()` adds a transition to an existing NC template. The transition must have “from” and “to” states and the condition must exist or the call fails. Three possible actions are supported: `unixcmd`, `condition`, and `mail`. If UNIX command or mail actions are passed, there must be a double-quoted `arg1`.

7.10.29 nci_transition_delete

```

Result
nci_transition_delete(
    NC_Defn &def,
    const char *from_state,
    const char *to_state,
    const char *condition,
    const char *action,
    const char *arg0,
    const char *arg1
);

```

The function `nci_transition_delete()` deletes a transition between two states.

7.10.30 nci_transition_find

```

NC_Transition
nci_transition_find(
    NC_Defn &def,
    const char *from_state,
    const char *to_state,
    const char *condition
);

```

The function `nci_transition_find()` gets the transition in NC template identified by *def* going from the state *from_state* to *to_state* on the condition *condition*. The function returns a valid `NC_Transition`, which is a handle on the Nerve Center state-transition, and returns a default (invalid) `NC_Transition` if there is an error.

7.10.31 nci_transition_get

```
NC_Transition
nci_transition_get(
    NC_State &from_state,
    NC_State &to_state,
    const char *condition
);
```

7.10.31.1 Description

The function `nci_transition_get()` gets the handle on transition from the state identified by *from_state* to the state identified by *to_state* on the condition *condition*. The function returns a valid `NC_Transition`, which is a handle on the Nerve Center state-transition, and returns a default (invalid) `NC_Transition` if there is an error.

7.10.31.2 Event Request Example

This program creates pollrates, severities, and conditions, then uses them to define a NC template for managing an SNMP host.

CODE EXAMPLE 7-1 Sample Event Request

```
#include hi.hh
#include stdlib.h
#include sys/types.h
#include "error.hh"
#include "nci/nc_api.hh"
#include "nci/nc_def.hh"
#include "nci/nc_coll.hh"

Error    error;

void create_pollrates();
void create_severities();
void create_conditions();
void create_template(char*);
void fail(const char*);

NC_Defn nc_handle;
```

CODE EXAMPLE 7-1 Sample Event Request (*Continued*)

```

main(int argc, char**argv)
{
    if (argc != 2) {
        printf("Usage: create_template template_name\n");
        exit(1);
    }
    DU error_msg;
    if (!nci_init("localhost", error_msg)) {
        printf("libnci initialization failed:%s/n", error_msg.chp());
        exit(1);
    }
    create_pollrates();
    create_severities();
    create_conditions();
    create_template(argv[1]);
}

void
create_pollrates()
{
    if (!nci_pollrate_add("FastPoll", 20))
        fail("nci_pollrate_add");
    if (!nci_pollrate_add("SlowPoll", 60))
        fail("nci_pollrate_add");
}

void
create_severities()
{
    if (!nci_severity_add("down", 18, "red"))
        fail("nci_severity_add");
    if (!nci_severity_add("ok", 16, "green"))
        fail("nci_severity_add");
}

void
create_conditions()
{
    const char *SYS = "$tmp = \"/internetClassId=\
{1 3 6 1 4 1 42 3 2 3 1 1 3 6 1 2 1 1 0}\";\n$pollfdn =
append_rdn($pollfdn,$tmp);\nTRUE;";

    if (!nci_condition_add("SetSystem", "Set the polling
fdn", SYS))
        fail("nci_condition_add");
}

```

CODE EXAMPLE 7-1 Sample Event Request (*Continued*)

```

        if (!nci_condition_add("IsSystemDescr", "Poll for System
Description",
        "defined(&sysDescr);"))
        fail("nci_condition_add");
        if (!nci_condition_add("IsNotSystemDescr",
        "If Can't Reach the System",
        "NOT defined(&sysDescr);"))
        fail("nci_condition_add");
        if (!nci_condition_add("UndefineSystemDescr",
        "Undefine the System
        Description", "undefine(&sysDescr);"))
        fail("nci_condition_add");
    }

void
create_template(
    char *name
)
{
    NC_Defn nc = nci_template_create(name, "Test Template");
    if (!nc)
        fail("nci_template_create");

    // Adding States
    if (!nci_state_add(nc, "Init", "Poll", "Normal",
        "Initialization State"))
        fail("nci_state_add");
    if (!nci_state_add(nc, "Poll", "Poll", "Normal",
        "Polling State"))
        fail("nci_state_add");
    if (!nci_state_add(nc, "Up", "Poll", "ok", "System Up"))
        fail("nci_state_add");
    if (!nci_state_add(nc, "Down", "Poll", "down", "System Down"))
        fail("nci_state_add");

    // Add Transitions
    if (!nci_transition_add(nc, "Init", "Poll", "SetSystem",
        NULL, NULL, NULL))
        fail("nci_transition_add");
    if (!nci_transition_add(nc, "Poll", "Up", "IsSystemDescr",
        "CONDITION", "UndefineSystemDescr", NULL))
        fail("nci_transition_add");
    if (!nci_transition_add(nc, "Poll", "Down",
        "IsNotSystemDescr", NULL, NULL, NULL))
        fail("nci_transition_add");
}

```

CODE EXAMPLE 7-1 Sample Event Request (*Continued*)

```

        if
(!nci_transition_add(nc,"Up","Down","IsNotSystemDescr",NULL,NULL
,NULL))
    fail("nci_transition_add");
    if (!nci_transition_add(nc,"Up","Up","IsSystemDescr",
        "CONDITION","UndefineSystemDescr",NULL))
fail("nci_transition_add");
    if (!nci_transition_add(nc,"Down","Up","IsSystemDescr",
        "CONDITION","UndefineSystemDescr",NULL))
fail("nci_transition_add");

        if (!nci_template_store(nc, name, "Def"))
fail("nci_template_store");
}

void
fail(const char *s)
{
    printf("%s: Failed exiting.\", s);
    exit(1);
}

```


Topology API

The Topology API is designed for use by Solstice Enterprise Manager (Solstice EM) application developers. This interface hides the topology implementation from application developers, allows applications to be more easily ported, and allows for faster development of topology-based applications.

This chapter comprises the following topics:

- Section 8.1 “Topology Classes” on page 8-2
- Section 8.2 “Class Overview” on page 8-5
- Section 8.4 “Persistent Object Classes” on page 8-10
- Section 8.5 “Utility Classes” on page 8-13
- Section 8.6 “Topology API Concepts” on page 8-14
- Section 8.7 “Examples” on page 8-15
- Section 8.8 “Class Reference” on page 8-29

Note – Some understanding of GDMO/ASN.1 and network management principles is required to effectively use the Topology API. For example, it is necessary to understand object identifiers (OID) and distinguished names (DN) and how these relate to the object registration tree and management information tree (MIT). In addition, a high-level understanding of the Solstice EM architecture and topology model is necessary.

8.1 Topology Classes

The Topology API consists of the following classes:

TABLE 8-1 Topology API Classes

Class	Description
EMStatus Class	Reports status, including error
EMIntegerSet Class	Implements a general purpose integer set
EMIntegerSetIterator Class	Provides a convenient method to visit each member of the integer set
EMTopoPlatform Class	Represents the Topology API as a whole
EMObject Class	Specifies the interface supported by all the persistent object classes
EMTopoNodeDn Class	Uniquely Identifies a topology node out of a set of topology node objects
EMTopoTypeDn Class	Uniquely Identifies a topology type out of the set of topology types
EMTopoNode Class	Represents a topology node
EMTopoType Class	Represents a topology type
EMAgent Class	Contains the agent interface common between EMCmipAgent, EMRpcAgent, and EMSnmpAgent
EMCmipAgentDn Class	Identifies one cmip agent object out of the set of cmip agent objects
EMCmipAgent Class	Represents the MIS object which contains configuration information
EMRpcAgentDn Class	Identifies one rpc agent object out of the set of rpc agent objects
EMRpcAgent Class	Represents the MIS object which contains configuration information
EMSnmpAgentDn Class	Identifies one snmp agent object out of the set of snmp agent objects
EMSnmpAgent Class	Represents the MIS object which contains configuration information

Application developers must work within the hierarchical model with containers and objects, but they do not need to understand the multiple objects within the MIS that represent individual topology elements.

The Topology API is not intended to provide access into non-topology related features provided by the PMI or Nerve Center Interface. Applications such as the Topology Import/Export Tool, Discover, and large parts of the Viewer should be achievable with this interface.

8.1.1 General Comments

- Standard RogueWave Tools.h++ classes and templates are used instead of the PMI's DataUnit, Morf, Array, and Queue classes. The RogueWave Tools.h++ classes are simple and easier to understand. Solstice EM 4.1 comes with Rouge Wave tools with standard IO Stream supplied with Forte Update 2 compiler. The Furthermore, many developers just starting to work with Solstice EM are already familiar with the Tools.h++ library.
- Fundamental concepts, such as accessing topology objects distributed across multiple MISs, and handling duplicate topology node names, have been factored into the design of the Topology API to help deal with these issues.
- In general, the number of lines of code needed to perform some operation on topology objects are fewer (sometimes many times fewer) with the Topology API versus the PMI. In addition, the code is readable and easy to maintain. This should allow for faster development of topology-based applications.

8.1.2 General Description

Using the Topology API, developers can create applications for the Solstice EM platform without learning the details of the MIT naming tree. The following figure gives an idea of how the Topology API is positioned in Solstice EM.

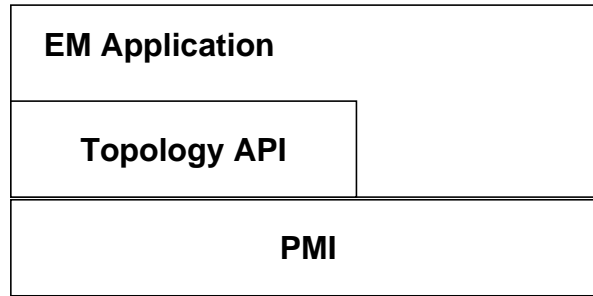


FIGURE 8-1 Position of the Topology API

8.2 Class Overview

Topology classes are related to the GDMO, the PMI, and persistent objects.

8.2.1 Relationship to the GDMO

In terms of the GDMO, the Topology API provides a concrete C++ interface to the MIT objects described in the table below.

TABLE 8-2 Topology API and GDMO Object Relationship

Topology API C++ Class	Objects in MIT
EMTopoNode Class	topoNode objects contained under topoNodeDBId=NULL topoView objects contained under topoViewDBId=NULL topoViewNode objects contained under topoViewDBId=NULL/ topoNodeId=XX
EMTopoType Class	topoType objects contained under topoTypeDBId=NULL
EMCmipAgent Class	cmipAgent objects contained under agentTableType="CMIP"
EMRpcAgent Class	rpcAgent objects contained under agentTableType="RPC"
EMSnmipAgent Class	cmipsnmipProxyAgent objects contained under internetClassId={ 1 3 6 1 4 1 42 2 2 2 9 2 4 1 0}

Through the C++ interface, the Topology API provides services that are equivalent to the following GDMO services:

- Create, delete, set attributes of, and get attributes of the object classes topoNode, topoView, topoViewNode, topoType, cmipsnmipProxyAgent, cmipAgent, and rpcAgent.
- Provide support for the actions topoGetNodeMODataByNodeList, topoNodeChildAttrByView, topoNodeGetByMOSet, topoNodeGetByName, topoNodeGetByType, topoNodeGetByMO, topoGetViewGraph, getTopoNodesAttributes, and setTopoNodesAttributes supported by the topoNodeDB object class.
- Provide support for the processes of objectCreation, objectDeletion, and attributeValueChange notifications for the topology object classes topoNode, topoView, topoViewNode.

8.2.2 Relationship to PMI

The Topology API is built on top of the PMI. If your client application only needs to manipulate topology nodes, topology types, cmip agents, rpc agents, and snmp agents, then the only places where the PMI must still be used are the following:

- A connection to an MIS, established using the PMI `Platform` class. After the platform instance has been successfully initialized, the Topology API is initialized by calling the `EMTopoPlatform::initialize` method with the `Platform` instance as a parameter.
- If the application supports access control application features, then `Platform::get_authorized_features()` must be used to find out which features a particular user is authorized to use.
- When the `Morf` class is used for setting or getting the `EMTopoNode::user_data` attribute. There is no way around this, since the `user_data` attribute can contain data defined by any ASN.1 syntax.

Of course, if the application needs to access additional objects in the MIT beyond those outlined in TABLE 8-2 on page 5, then the PMI `Image` class must be used. As an example, the Solstice EM Network Viewer makes use of the Topology API to access topology nodes and topology types, the NCI API to manipulate requests, and the PMI for connection, access control features, and a few miscellaneous operations.

The following restrictions on usage of the PMI are necessary in order for the Topology API to function correctly:

- If `Platform::replace_discriminator()`, or `Platform::replace_discriminator_classes` eliminate the sending of unwanted events from the MIS, the discriminator must allow all events for the following object classes: `topoNode`, `topoView`, `topoViewNode`, `topoType`, `cmipAgent`, `rpcAgent`, and `cmipsnmpProxyAgent`.

For example, an application that doesn't subscribe to any events itself, that is, doesn't use `Album::when()`, `Image::when()`, or `Platform::when()`, and calls `Platform::replace_discriminator()` with the argument "or : {}", "which instructs the MIS not to send any events to the application, eliminates unnecessary event traffic and processing time.

However, that same application using the Topology API, would require the following syntax.

```
#include pmi/hi.hh

Array(DU) object_classes;
object_classes.alloc(7);
object_classes[0] = "topoNode";
object_classes[1] = "topoView";
object_classes[2] = "topoViewNode";
object_classes[3] = "topoType";
object_classes[4] = "cmipAgent";
object_classes[5] = "rpcAgent";
object_classes[6] = "cmipsnmpProxyAgent";

Platform::replace_discriminator_classes(object_classes)
```

- Platform::set_attr_coder() should not be called to change the encoder/decoder for any of the GDMO attributes of the GDMO object classes topoNode, topoView, topoViewNode, topoType, cmipAgent, rpcAgent, or cmipsnmpProxyAgent.

8.3 EMTopoPlatform Class

The EMTopoPlatform class represents the Topology API as a whole. Only one instance of the EMTopoPlatform class is allowed. This instance is initialized by calling EMTopoPlatform::initialize(), and is accessed through the EMTopoPlatform::instance() method¹. The EMTopoPlatform class provides various methods, or member functions including:

- Get all MIS systems reachable from the connected MIS
- Find topology nodes by name, type, or managed object
- Find CMIP, RPC, and SNMP agents by managed object.
- Get the topology pathname(s) by topology node DN.

1. For those familiar with C++/OO design, the EMTopoPlatform class uses the Singleton pattern.

```

#include pmi/hi.hh
#include topo_api/topo_api.hh

Platform platform;

if (!platform.connect("mishost", "em_client")) {
    cerr << "Failed to connect to " << "mishost" << endl;
    exit(-1);
}
EMTopoPlatform::initialize(platform);

```

The above code shows how the Topology API is initialized.

8.3.1 get_attributes_by_mo()

```

EMStatus get_attributes_by_mo(
    const RWTValSlistRWCString& managed_objects,
    const EMIntegerSet& attributes,
    RWTValSlistEMTopoNode& nodes,
    const RWTValSlistRWCString&
        system_names = RWTValSlistRWCString()
)

```

This method enables you to get attribute values of topology nodes that represent the given managed objects. To use this method, you must specify the following parameters:

- *managed_objects*

A list of the managed objects.

- *attributes*

A list of attributes whose values you want to get. If you provide an empty list of attributes, this method gets all the attribute values of the topology nodes. In either case, *attributes* should be created with `EMTopoNode::num_attributes`.

If this method completes successfully, it returns a list of topology nodes *nodes*. Each node in the returned list contains at least one of the given *managed_objects* in the node's `topoNodeMOSet` attribute. Each node in the list contains the values of the given attributes.

Because a topology node can contain more than one managed object, it is possible that the returned list of topology nodes contains a number of nodes that is less than the number of the given managed objects. Because a managed object can be contained by multiple topology nodes, it is possible that the returned list of topology nodes contains a number of nodes that is greater than the number of the given managed objects.

Note – For topology view-related attributes, it is the user's responsibility to check whether a node is a topology view before getting its attribute values.

8.3.2 `set_attributes_by_mo()`

```
EMStatus set_attributes_by_mo(
    const RWTValSlistRWCString& managed_objects,
    const EMIntegerSet& attributes,
    const EMTopoNode &reference_node,
    RWTValSlistEMTopoNode& nodes,
    const RWTValSlistRWCString&
        system_names = RWTValSlistRWCString()
)
```

This method enables you to set attribute values of topology nodes that represent the given managed objects. To use this method, you must specify the following parameters:

- *managed_objects*
A list of the managed objects.
- *attributes*
A list of attributes whose value you want to set.
- *reference_node*
EMTopoNode object filled with the values for *attributes*.

If this method completes successfully, it returns a list of topology nodes *nodes*. Each node in the returned list contains, at least, one of the given *managed_objects* in the node's `topoNodeMOSet` attribute. Each node in the list contains the values of the given attributes as specified in the reference node.

Because a topology node can contain more than one managed object, it is possible that the returned list of topology nodes contains a number of nodes that is less than the number of the given managed objects. Because a managed object can be contained by multiple topology nodes, it is possible that the returned list of topology nodes contains a number of nodes that is greater than the number of the given managed objects.

Note – For topology view-related attributes, it is the user's responsibility to check whether a node is a topology view before getting its attribute values.

8.4 Persistent Object Classes

The Topology API provides an interface to five Persistent Object Class (POC) objects in the MIS: topology nodes, topology types, SNMP agents, CMIP agents, and RPC agents. Unlike the PMI, where the `Image` class provides a generic interface to any persistent object in the MIS, the Topology API provides a concrete, type-safe C++ class for each of these five classes.

8.4.1 EMOBJECT Class

The `EMObject` class is an abstract base class from which the concrete POC classes are derived. This class declares the common methods that all POC classes support. The common methods include:

- Creation and deletion in the MIS's persistent store
- Loading/storing attributes from/to the MIS

Other methods common among POC classes are not declared in the `EMObject` class because the signature of the method isn't exactly the same. For example, all the POC classes support the `compare_all_attributes()` and `compare_some_attributes()` methods.

8.4.2 EMOBJECT Member Functions

The following are member functions of the EMOBJECT Class.

compare_all_attributes

```
RWBoolean
EMTopoNode::compare_all_attributes(const EMTopoNode& peer )
```

compare_some_attributes

```
RWBoolean
EMTopoNode::compare_some_attributes(const EMTopoNode& peer )
```

Because the type of the *peer* parameter differs for each POC, the methods cannot be included in the EMOBJECT base class. Each POC class also provides additional methods specific to the POC; in particular, access methods to the attributes of the POC are provided.

8.4.3 EMTOPOTYPE Class

An instance of the EMTopoType class represents a topology type. Every topology node is classified as a particular topology type. The topology types form a hierarchy with the seven base types “Array,” “Bus,” “Container,” “Device,” “Monitor,” “Link,” and “Sun” with other subtypes derived from them.

Beyond the standard POC methods, which allow you to create, delete, and compare topology types, the EMTopoType class provides the static methods `is_array()`, `is_bus()`, `is_container()`, `is_device()`, `is_monitor()`, `is_link()`, and `is_view()` which can be used to categorize topology types.

8.4.4 EMTopoNode Class

The `EMTopoNode` class represents a topology node, which is the unit of management in Solstice EM. Using the standard POC methods, you can create, delete, and compare topology nodes. Using the `EMTopoNode` access methods, you can get and set the name, topology pathname, logical and geographical location, topology type, and associated managed objects and their corresponding CMIP, RPC, and/or SNMP agent objects among others attributes. The `EMTopoNode` class also provides a callback mechanism to notify clients when a topology node has been created, deleted, or has had one or more attributes changed.

8.4.5 EMSnmpAgent Class

An instance of the `EMSnmpAgent` class represents the MIS object that contains configuration information for an SNMP agent. The configuration information includes such information as the read and write community strings, supported MIBs, and transport address.

Note – This class does not provide an interface to the agent's managed objects, only to Solstice EM's configuration information for the agent.

8.4.6 EMCmipAgent Class

An instance of the `EMCmipAgent` class represents the MIS object that contains configuration information for a CMIP agent. The configuration information includes the CMIP MPA hostname and port number, list of managed objects DNs, network SAP, transport selector, presentation selector, session selector, and application entity title (AET).

Note – This class does not provide an interface to the agent's managed objects, only to Solstice EM's configuration information for the agent.

8.4.7 EMRpcAgent Class

An instance of the EMRpcAgent class represents the MIS object that contains configuration information for an RPC agent. The configuration information includes the read and write community strings, and supported schemas.

Note – This class does not provide an interface to the agent's managed objects, only to Solstice EM's configuration information for the agent.

8.5 Utility Classes

8.5.1 EMIntegerSet Class

The EMIntegerSet class implements a general-purpose integer set over the numbers 0 to n . It is used in the Topology API to communicate which attributes of a POC an API method should operate on.

8.5.2 EMStatus Class

Instances of class EMStatus are returned by almost every API method to report status, including errors. A conversion operator to RWBoolean is provided so that EMStatus can be evaluated in boolean expressions. A value of FALSE means there was an error, otherwise success.

8.6 Topology API Concepts

8.6.1 Element Naming

Applications must be able to access individual topology elements without traversing the entire topology hierarchy. The mapping of topology element names to that of a file system model for unique naming is supported. In the event that a file system style reference is ambiguous within the underlying MIT, the method invoked fails and reports the appropriate error. As an example of this naming, an element named “Parrothead,” located under the “Internet” view, located under the “Root” view, would be referenced as `/Root/Internet/Parrothead`. There can be only one root, and as such, the root is represented as `/` within this model.

8.6.2 Duplicate Topology Node Names

The administrative names of the `EMTopoType`, `EMCmipAgent`, `EMSnmppAgent`, and `EMRpcAgent` persistent objects are guaranteed to be unique. In contrast, the administrative name of the `EMTopoNode` is not guaranteed to be unique.

To address this, the `EMTopoPlatform` class provides several methods to return a list of `EMTopoNodeDn` instances that:

- Have the same administrative name
- Have the same type
- Share the same proxy agent object
- Share the same managed object Dn's

8.6.3 MIS-MIS Awareness

Each persistent object class supports access to any object instance visible from the connected MIS. For example, if MIS A and MIS B have a 2-way MIS-MIS connection setup, you can connect to MIS A, then modify `EMTopoNodes`, `EMTopoTypes`, and so forth, on MIS B.

The `EMTopoPlatform` find methods, such as `find_nodes_by_name()`, and `find_nodes_by_type()`, perform the search on the entire set of objects visible from the connected MIS. Using the above example of MIS A and MIS B, if you connect to MIS A, and `find_nodes_by_type()`, you see a list of all `EMTopoNodeDn`'s on MIS A and/or MIS B of the indicated type.

8.6.4 Performance Considerations

Because the Topo API is built on top of the PMI, most operations take slightly longer when using the Topo API versus writing the code directly with PMI.

In terms of memory usage, however, the persistent object classes require much less memory cache information about an object than if an `Image` class had been used instead. This is because the Topo API classes can optimize the data storage; they know exactly what attributes each managed object contains.

8.7 Examples

This section presents several examples showing how to use the Topology API for common tasks.

8.7.1 Makefile

The following `Makefile` was used to compile all of the programs in this section. The version of the SUN C++ SparcCompiler used is "SC4.0 18 Oct 1995 C++ 4.1". (This is the output from "CC -V"). This `Makefile` and the following sample programs can be found in `$EM_HOME/src/topo_api` directory.

```
CCFLAGS = +w -g -noex -I${EM_HOME}/include
-I${EM_HOME}/include/pmi
LDFLAGS = -L${EM_HOME}/lib -ltopo_api -lpmi -lrwtool -lsched
-lns1 -lsocket -lgen -R/opt/SUNWconn/em/lib

EXES =print_topo topo_events traverse
OBS =$(EXES:%=%.o)

all: $(EXES)

print_topo: print_topo.o
    $(LINK.C) -o $@ $@.o

topo_events: topo_events.o
    $(LINK.C) -o $@ $@.o

traverse: traverse.o
    $(LINK.C) -o $@ $@.o

clean:
    rm -rf $(EXES) $(OBS) Templates.DB;
```

8.7.2 Finding Topology Nodes

This program accepts as input the name of a topology node. The program then uses `EMTopoPlatform::find_nodes_by_name()` to find all topology nodes with the given name. Then some information for each node is displayed. This program highlights the fact that more than one topology node can have the same name, so a Solstice EM client should never assume that the topology node names are unique. That is why the Topology API uses instances of `EMTopoNodeDn` to uniquely identify a single topology node.

```

#include <stdio.h>
#include topo_api/topo_api.hh

int
main(int argc, char**argv)
{
    if (argc < 2) {
        cerr << "Usage: " << argv[0] << " node-name" << endl;
        exit(-1);
    }
    RWCString node_name = argv[1];

    Platform platform(duEM);

    if (!platform.connect("", "em_sample")) {
        cerr << "Couldn't connect!" << endl;
        exit (-2);
    }

    EMTopoPlatform::initialize(platform);

    RWTValSlistEMTopoNodeDn nodes;
    EMTopoPlatform::instance()-
    >find_nodes_by_name(node_name, nodes);

    if (nodes.isEmpty()) {
        cerr << "No Topology Node Named " << node_name << endl;
        exit(-3);
    }

    for (RWTValSlistIteratorEMTopoNodeDn i(nodes); i(); ) {
        EMTopoNode node(i.key());
        EMStatus status;

        if (!(status = node.load_all_attributes())) {
            cerr << "Error: " << status << endl;
            exit(-4);
        }
    }
}

```

```
//
// The stream output operator << is defined for
// EMTopoNode, EMTopoType, EMCmipAgent, EMSnmpAgent,
// EMRpcAgent, providing an easy way to print out
// the values of an objects while debugging.
cout << "-----debug output-----" << endl;
cout << node << endl;
cout << "-----debug output-----" << endl;

//
// Normally, you want to do more with the values than
// print them out.
//
EMTopoNode::Severity severity;
node.get_severity(severity);

RWCString name;

node.get_name(name); // named the same as node_name

RWCString type_name;
node.get_type_name(type_name);

EMTopoNode::GeoLocation geographical_location;
RWBoolean is_geographical_location_null;
node.get_geographical_location(geographical_location,
                               is_geographical_location_null);

RWTValslistEMCmipAgentDn cmip_agents;
RWTValslistEMSnmpAgentDn snmp_agents;
RWTValslistEMRpcAgentDn rpc_agents;
node.get_cmip_agents(cmip_agents);
node.get_snmp_agents(snmp_agents);

node.get_rpc_agents(rpc_agents);

cout << "Node named " << name << " is of type "
<< type_name << endl
      << "The most severe outstanding alarm is " << severity
<< "." << endl
```



```

        << "The node is located at ";
    if (is_geographical_location_null)
        cout << "unknown";
    else
        cout << geographical_location;
    cout << " in the world" << endl;

    cout << "CMIP agents: ";
    if (cmip_agents.isEmpty()) {
        cout << "none" << endl;
    } else {
        cout << endl;
        for
(RWTValSlistIteratorEMCmipAgentDn j(cmip_agents); j(); ) {
            cout << "\t" << j.key() << endl;
        }
    }

    cout << "RPC agents: ";
    if (rpc_agents.isEmpty()) {
        cout << "none" << endl;
    } else {
        cout << endl;
        for
(RWTValSlistIteratorEMRpcAgentDn j(rpc_agents); j(); ) {
            cout << "\t" << j.key() << endl;
        }
    }

    cout << "SNMP agents: ";
    if (snmp_agents.isEmpty()) {
        cout << "none" << endl;
    } else {
        cout << endl;
        for
(RWTValSlistIteratorEMSnmpAgentDn j(snmp_agents); j(); ) {
            cout << "\t" << j.key() << endl;
        }
    }
}
return 0;
}

```

8.7.3 Registering Events for `EMTopoNode`

The `EMTopoNode` class provides an event subscription service to notify clients when a topology node is created, deleted, or modified. This service is not offered by the other persistent object classes.

The following program registers for all three types of events, then proceeds to create, modify, and destroy a single topology node in order to cause some events to be sent to the registered callback. A description of each event is printed to `stdout`.

```

#include <stdio.h>
#include topo_api/topo_api.hh

void topo_event_cb(
    const EMTopoNodeCallbackData& cbd
);

int
main(int /*argc*/, char** /*argv*/)
{
    Platform platform(duEM);

    if (!platform.connect("", "em_sample")) {
        cerr << "Couldn't connect!" << endl;
        exit (-2);
    }

    EMTopoPlatform::initialize(platform);

    //
    // Register for create, delete, and attribute change events
    // on EMTopoNode objects.
    //

    EMTopoNode::register_callback(em_any_event, topo_event_cb, NULL);

    //
    // Now we will create, modify, and then delete an EMTopoNode
    // to trigger some events.
    //
    //
    // Find the root node(s) (there will one for each MIS)
    // so we have a parent view to create a topology node in.
    //
    RWTValSlistEMTopoNodeDn roots;
    EMTopoPlatform::instance()->find_root_nodes(roots);
}

```

```
// Okay, now we have to set the three mandatory attributes
// for creating a topology node: name, type_name, and parents.
//
EMTopoNode node;

node.set_name("first-name");

node.set_type_name(EMTopoTypeDn::host);
//
// Arbitrarily use the first root node in the list as the
// parent of the new topology node.
//
node.add_parent(roots.first());

//
// Create the node
//

EMStatus status;
if (!(status = node.create_with_all_attributes())) {
cerr << "Error: " << status << endl;
exit(-1);
}

//
// After the create has completed, the EMTopoNode::dn
// attribute, the unique identifier, is set.
//
EMTopoNodeDn node_dn;
node.get_dn(node_dn);
cout << "Created Topology Node Id=" << node_dn
<< " with parent Id=" << roots.first() << endl;

//
// Modify some attributes
//
// We don't want to store the parents and type_name attributes
// again, so we need to reset the EMTopoNode object.
node.clear_all_attributes();
```

```

//
// EMTopoNode::dn is the only mandatory attribute when doing
// a store or load.
//
node.set_dn(node_dn);
node.set_name("second-name");
node.set_logical_location(roots.first(),Location(/*x=*/12,/*y=*/34,/*z=*/56));
node.set_geographical_location(GeoLocation(/*longitude=*/-112.0,/*latitude=*/45.0));

//
// store_all_attributes() only stores attributes that
// have been set.
//
if (!(status = node.store_all_attributes())) {
cerr << "Error: " << status << endl;
exit(-2);
}

//
// destroy the node
//
if (!(status = node.destroy())) {
cerr << "Error: " << status << endl;
exit(-3);
}
}

void topo_event_cb(
    const EMTopoNodeCallbackData& cbd
)
{
    switch(cbd.event_type) {
        case em_create_event:
            cout << "topo_event_cb: node " << cbd.node_dn << " created"
<< endl;
            break;
        case em_delete_event:

```

```

        cout << "topo_event_cb: node " << cbd.node_dn
<< " deleted" << endl;
        break;
        case em_change_event:
            cout << "topo_event_cb: node " << cbd.node_dn
<< " modified" << endl;

        //
        // cbd.changes is an instance of EMTopoNode contains all
        // the changes.
        //
        EMIntegerSet attributes(EMTopoNode::num_attributes);
        cbd.changes.get_active_attributes(attributes);
        cout << "\tAttributes changed: ";
        for (EMIntegerSetIterator i(attributes); i.next(); ) {
            cout << EMTopoNode::get_attribute_name(i.member());
        }
        cout << endl;

        for (i.reset(); i.next(); ) {
            switch (i.member()) {
                case EMTopoNode::name:
                {
                    RWCString name;
                    cbd.changes.get_name(name);
                    cout << "\tname changed to " << name << endl;
                }
                break;
                case EMTopoNode::logical_locations:
                {
                    RWTValSlistLocationInParent locations;
                    cbd.changes.get_logical_locations(locations);
                    cout << "\tlogical_location in parent view " <<
locations.first().parent <<
                    " is " << locations.first().location
<< endl;
                }
                case EMTopoNode::children:

```

```

        {
            RWTValSlistEMTopoNodeDn children;
            cbd.changes.get_children(children);

            cout << "\tchildren changed to {"<
            for
(RWTValSlistIteratorEMTopoNodeDn j(children); j(); ) {
                cout << j.key() << " ";
            }
            cout << "}" << endl;
        }
        break;
    }
}
}

```

8.7.4 Printing the Topology Hierarchy

The topology hierarchy forms a directed acyclic graph. It is a graph, rather than a tree, because each topology node except the root node can have more than one parent. It is acyclic because the parent-child relationship should not have any loops. The following program traverses the topology depth-first, starting at the root node(s). As each node is visited, the `EMTopoNode::topology_pathnames` attribute is printed to `stdout`.

Note that the `cache_view_graph` option of `EMTopoPlatform::initialize()` is set to `TRUE` this time since the program accesses the `EMTopoNode::topology_pathnames` attribute of every node.

```
#include <stdio.h>
#include <iostream.h>
#include topo_api/topo_api.hh

void traverse(
    const EMTopoNodeDn& dn
);

long num_traversed = 0;

int
main(int /*argc*/, char** /*argv*/)
{
    Platform platform(duEM);

    if (!platform.connect("localhost","em_sample")) {
        cerr << "Couldn't connect!" << endl;
        exit (-1);
    }

    EMTopoPlatform::initialize(platform,TRUE);

    EMStatus status;

    //
    // Find the root node(s) (there will one for each MIS)
    // so we have a parent view to create a topology node in.
    //
    RWTValSlistEMTopoNodeDn roots;
    if (!(status = EMTopoPlatform::instance()->
find_root_nodes(roots))) {
        cerr << status << endl;
        exit(-2);
    }
}
```



```

    // Traverse the topology of each MIS
    //
    for (RWTValSlistIteratorEMTopoNodeDn i(roots); i(); ) {
        traverse(i.key());
    }

    //
    // Note, the number of nodes traversed is most likely not equal
    // to the number of nodes since a node with n parents would be
    // traversed n times with the simple algorithm used.
    //
    cout << "Num Nodes Traversed = " << num_traversed << endl;
}

void
traverse(
    const EMTopoNodeDn& dn
)
{
    EMStatus status;

    EMTopoNode node(dn);

    num_traversed++;

    //
    // Load the topology pathnames of the topology node, and also
    // the children if the topology node is a view (container or
    // monitor type)
    //
    EMIntegerSet attributes(EMTopoNode::num_attributes);
    attributes.add(EMTopoNode::topology_pathnames);

    // We need the children attribute so that we can continue
    // or depth-first traversal.
    if (EMTopoPlatform::instance()->is_view(dn)) {
        attributes.add(EMTopoNode::children);
    }
}

```

```

if (!(status = node.load_some_attributes(attributes))) {
    cerr << "Error: " << status << endl;
    exit(-3);
}

//
// Print out all the possible topology pathnames for the node.
// A node may have more than one valid pathname from the root
// node because this is a directed acyclic graph not a tree.
//

RWTValSlistRWCString topology_pathnames;
if (!(status =
node.get_topology_pathnames(topology_pathnames))) {
    cerr << "Error: " << status << endl;
    exit(-3);
}

cout << "{ ";
for (int i = 0; i < topology_pathnames.entries(); i++) {
    cout << topology_pathnames[i];
    if (i != topology_pathnames.entries() - 1)
        cout << ", ";
}
cout << "}" << endl;

//
// Recur on the node's children (if it has any)
//
if (EMTopoPlatform::instance()->is_view(dn)) {
    RWTValSlistEMTopoNodeDn children;
    if (!(status = node.get_children(children))) {
        cerr << "Error: " << status << endl;
    }
    for (RWTValSlistIteratorEMTopoNodeDn j(children); j();) {
        traverse(j.key());
    }
}
}

```

8.8 Class Reference

This section includes the following classes:

- EMStatus Class
- EMIntegerSet Class
- EMIntegerSetIterator Class
- EMTopoPlatform Class
- EMObject Class
- EMTopoNodeDn Class
- EMTopoTypeDn Class
- EMTopoType Class
- EMAgent Class
- EMCmipAgentDn Class
- EMCmipAgent Class
- EMSnmpAgentDn Class
- EMRpcAgent Class
- EMSnmpAgentDn Class
- EMSnmpAgent Class

8.9 EMStatus Class

Inheritance: none

```
#include topo_api/topo_api.hh
```

Instances of class EMStatus are returned by almost every API method to report status, including errors. A conversion operator to RWBoolean is provided so that EMStatus can be evaluated in Boolean expressions. A value of FALSE means there was an error, otherwise success. The following sample code shows the basic usage.

Static Variables

```
static EMStatus EMStatus::success;
```

This static public member can be used to compare to instances of EMStatus. If they are equal, then the operation succeeded. For example:

```
if (EMStatus::success == node.load_all_attributes()) {
    cout << "succeeded" << endl;
}
```

Enum

```
enum EMStatus::Code {
    successful,
    pmi_error,
    object_doesnt_exist,
    attribute_is_not_creatable,
    attribute_is_not_storeable,
    attribute_not_set,
    key_not_found,
    missing_mandatory_attribute,
    cannot_set_attribute,
    decode_error,
    encode_error,
    attribute_not_registered,
    does_not_exist,
    already_exists,
    invalid_arg,
    not_implemented,
    not_supported,
    view_graph_not_cached,
    duplicate_cmip_managed_fdns,
    unknown_error, /* this means an internal error*/
    num_status
};
```

This is a list of all the possible statuses that can be returned by the API.

8.9.1 Constructors and Destructor

```
EMStatus();
```

```
EMStatus(Code error_code,  
         const RWCString& text);
```

```
EMStatus(const EMStatus& status);
```

```
~EMStatus();
```

Normally, only the default constructor is used by clients.

8.9.2 Operators

```
operator RWBoolean () const;
```

The following operator overloads are used for equality and logical equivalence.

```
EMStatus& operator =(const EMStatus& status);
```

```
RWBoolean operator ==(  
    const EMStatus& status  
) const;
```

And, the not-operator

```
RWBoolean operator !=(  
    const EMStatus& status  
) const;
```

The RWBoolean conversion operator equates `EMStatus::successful` with `TRUE`, otherwise `FALSE`.

Here is an example of how this can be used to test the return status of a method:

```
EMStatus status;
if (!(status = node.load_all_attributes())) {
    cerr << "Error: " << status << endl;
}
```

```
EMStatus::Code code() const;
```

Finally, it returns the status code.

8.9.3 Global Operators

```
ostream& operator<<(
    ostream& s,
    const EMStatus& status
);
```

The stream output operator << is defined to provide an easy way to print out the value of EMStatus.

8.10 EMIntegerSet Class

Inheritance: none

```
#include topo_api/topo_api.hh
```

The EMIntegerSet class implements a general-purpose integer set over the numbers 0 to n. It is used in the Topology API to communicate which attributes of a POC an API method should operate on. The example below shows how to load the name, topology type, and parents of a topology node.

8.10.1 Example

Instances of `EMIntegerSet` are used when you only want to load, store, or compare a subset of the attributes of one of the persistent object classes.

```
EMIntegerSet attrs(EMTopoNode::num_attributes);
attrs.add(EMTopoNode::name);
attrs.add(EMTopoNode::type_name);
attrs.add(EMTopoNode::parents);

node.load_some_attributes(attrs);
```

8.10.2 Constructors and Destructor

```
EMIntegerSet ();

EMIntegerSet(
    long n
);

EMIntegerSet(
    long n,
    RWBoolean initVal
);

~EMIntegerSet ();
```

`EMIntegerSet(n)` constructs a set of integers drawn from the universe numbered from 0 to *n*-1. By default, no numbers in the universe belong to the set. `EMIntegerSet(n,TRUE)` is the same except that every integer is a member of the set.

8.10.3 Operators

The following operator overloads are used for equality and logical equivalence.

```
EMIntegerSet& operator = (  
    RWBoolean b  
);
```

```
RWBoolean operator == (  
    const EMIntegerSet& set  
) const;
```

And, the *not*-operator,

```
RWBoolean operator != (  
    const EMIntegerSet& set  
) const;
```

Two EMIntegerSet instances are considered equal if they are both of the same dimension and have the exact same members.

```
RWBoolean operator [] (  
    long number  
) const;
```

Returns TRUE if the integer *number* is a member of the set.

The following performs a member-wise-and, exclusive-or, or Boolean operation on the specified set in comparison with another set of integers that must be of the same dimension.

```
EMIntegerSet& operator&=(  
    const EMIntegerSet& set  
);  
  
EMIntegerSet& operator^=(  
    const EMIntegerSet& set  
);  
  
EMIntegerSet& operator|=(  
    const EMIntegerSet& set  
);
```

8.10.4 Member Functions

add

```
void add(long number);
```

Adds integers to the set.

remove

```
void remove(long number);
```

Removes integers from the set.

is_member

```
void RWBoolean is_member(long number) const;
```

Returns TRUE if *number* is a member of the set.

num_members

```
void num_members(long num_number);
```

Returns the number of integers in the set.

max_members

```
long max_members() const;
```

Returns the number of integers in the universe of potential members.

resize

```
void resize(long n);
```

Resizes the integer set to the universe of integers 0 to n-1.

8.10.5 Global Operators

```

EMIntegerSet operator!(
    const EMIntegerSet& set
);

EMIntegerSet operator&(
    const EMIntegerSet& set1,
    const EMIntegerSet& set2
);

EMIntegerSet operator^(
    const EMIntegerSet& set1,
    const EMIntegerSet& set2
);

EMIntegerSet operator|(
    const EMIntegerSet& set1,
    const EMIntegerSet& set2
);

```

The `operator!` returns the member-wise negation of the input set. The other operator functions return the member-wise-and, exclusive-or, and-or, of two sets. Note that for the binary operations, the two sets must be of the same dimensions.

8.11 EMIntegerSetIterator Class

Inheritance: none

```
#include topo_api/topo_api.hh
```

The `EMIntegerSetIterator` class provides a convenient method to visit each member of the integer set.

8.11.1 Example

```
EMTopoNode node;
EMIntegerSet(EMTopoNode::num_attributes);
node.get_active_attributes(attrs);
for (EMIntegerSetIterator i(attrs); i(); ) {
    switch (i.key()) {
        case EMTopoNode::name:
            break;
        case EMTopoNode::type_name:
            break;
        case EMTopoNode::managed_objects:
            break;
        default:
            cout << "Some other attribute" << endl;
            break;
    }
}
```

8.11.2 Constructors and Destructor

```
EMIntegerSetIterator(const EMIntegerSet& set);
```

The Iterator will visit the members of *set* in numerical order.

And the corresponding destructor:

```
~EMIntegerSetIterator(const EMIntegerSet& set);
```

8.11.3 Member Functions

next

```
RWBoolean next();
```

`next()` advances the iterator one position and returns TRUE if the new position is valid, FALSE otherwise.

member

```
long member() const;
```

`member()` returns the integer member currently being visited.

reset

```
void reset();
```

`reset()` resets the iterator to the first integer member in the set.

8.12 EMTopoPlatform Class

Inheritance: none

```
#include topo_api/topo_api.hh
```

The `EMTopoPlatform` class represents the Topology API as a whole. Only one instance of the `EMTopoPlatform` class is allowed. This instance is initialized by calling `EMTopoPlatform::initialize()`, and is accessed through the `EMTopoPlatform::instance()` method.

The EMTopoPlatform class provides various methods, including:

- Get all MIS systems reachable from the connected MIS
- Find topology nodes by name, type, or managed object
- Find CMIP, RPC, and SNMP agents by managed object
- Get the topology pathname(s) by topology node DN

8.12.1 Example

```
#include pmi/hi.hh
#include topo_api/topo_api.hh

Platform platform;
if (!platform.connect("mishost", "em_client")) {
    cerr << "Failed to connect to " << "mishost" << endl;
    exit(-1);
}
EMTopoPlatform::initialize(platform);

EMStatus status;
RWTValSlistEMTopoNodeDn root_nodes;
if (!(status = EMTopoPlatform::instance()->
        find_root_nodes(root_nodes))) {
    cerr << "Error: " << status << endl;
    exit(-1);
}
```

The above code shows how the Topology API is initialized, and how the root nodes of the network topology are retrieved.

8.12.2 Static Member Functions

initialize

```
static RWBoolean initialize(  
    Platform& platform,  
    RWBoolean cache_view_graph = FALSE  
);
```

`initialize()` must be called before using any of the Topology API classes. This method should only be called after the *platform* has been successfully initialized. It returns `TRUE` on success, `FALSE` otherwise. The optional parameter *cache_view_graph* specifies whether you should optimize methods that operate over the topology view hierarchy.

If *cache_view_graph* is `TRUE`, then the topology view hierarchy will be cached into memory from the MIS using a special GDMO action `topoGetViewGraph` on the `topoNodeDBId=NULL` object. This optimization greatly increases the speed of loading `EMTopoNode::view_children` and `EMTopoNode::topology_pathnames` and executing `EMTopoPlatform::is_view()`. However, the view cache can require a significant amount of time and memory for large topology view hierarchy (> 5000 nodes).

instance

```
static EMTopoPlatform* instance();
```

After `initialize()` succeeds, the static method `instance()` returns a pointer to the `EMTopoPlatform` instance from which the non-static `EMTopoPlatform` methods can be invoked.

TABLE 8-3 summarizes when the *cache_view_graph* option should be turned on and off.

TABLE 8-3 *cache_view_graph* Option

cache_view_graph	application type
TRUE	Frequently calls EMTopoPlatform::is_view(), EMTopoPlatform::view_topology_pathnames(), EMTopoPlatform::topology_pathnames(), and/or loads EMTopoNode::view_children, EMTopoNode::topology_pathnames attributes.
FALSE	Does not use the above features or uses them infrequently. Client application should be as lightweight as possible and start up fast.

8.12.3 Access Member Functions

system_names

```
RWTValSlistRWCString system_names() const;
Platform& platform();
```

system_names() returns a list of all MIS names visible through the connection to the local MIS, including the local MIS name .

local_system_name

```
const RWCString& local_system_name() const;
Platform& platform();
```

local_system_name() returns the name of the MIS that the client application is connected to.

Note – For a remote MIS to be visible to client application, MMC (MIS-MIS Communication) must be set up between the local MIS and each remote MIS. This can be accomplished using the em_mismgr application.

8.12.4 General Member Functions

find_root_nodes

```
EMStatus find_root_nodes(
    RWTValSlistEMTopoNodeDn& root_nodes,
    const RWTValSlistRWCString& system_names =
        RWTValSlist RWCString()
) const;
```

Returns a list of all nodes named “Root” visible through the connection to the local MIS. The optional parameter *system_names* specifies the list of MISs to restrict the query. The MIS names in *system_names* should all appear in *system_names()*, otherwise *EMStatus::invalid_arg* will result. If *system_names* is empty (the default), then the list returned by *system_names()* is used.

find_nodes_by_name

```
EMStatus find_nodes_by_name(
    const RWCString& name,
    RWTValSlistEMTopoNodeDn& nodes,
    const RWTValSlistRWCString& system_names =
        RWTValSlist RWCString()
) const;
```

Same as *find_root_nodes()* except that all nodes named *name* are returned instead of all nodes named “Root.”

find_nodes_by_type

```
EMStatus find_nodes_by_type(
    const RWCString& type_name,
    RWTValSlistEMTopoNodeDn& nodes,
    const RWTValSlistRWCString& system_names =
        RWTValSlist RWCString()
) const;
```

Same as *find_root_nodes()* except that all nodes of type *type_name* are returned instead of all nodes named “Root.”

`find_nodes_by_managed_object`

```
EMStatus find_nodes_by_managed_object(
    const RWCString& managed_object,
    RWTValSlistEMTopoNodeDn& nodes,
    const RWTValSlistRWCString& system_names =
        RWTValSlist RWCString()
) const;
```

Same as `find_root_nodes()` except that all nodes that have *managed_object* listed in their `EMTopoNode::managed_objects` attribute are returned instead of all nodes named “Root”.

`is_view`

```
RWBoolean is_view(
    const EMTopoNodeDn& node_dn
) const;
```

Returns TRUE if the topology type of node *node_dn* is a ‘view’ type, that is, a subtype of `EMTopoTypeDn::container` or `EMTopoTypeDn::monitor`.

`view_topology_pathnames`

```
EMStatus view_topology_pathnames(
    const EMTopoNodeDn& view_dn,
    RWTValSlistRWCString& pathnames
) const;
```

Returns in *pathnames* a list of all topology pathnames for the node *view*. The node *view* should be a ‘view’, that is, a subtype of `EMTopoType::container` or `EMTopoTypeDn::monitor`. If *cache_view_graph* optimization is turned on, then this method is relatively inexpensive since no information needs to be retrieved from the MIS. At a minimum, *pathnames* will contain one pathname for each parent. However, since each parent can also have more than one parent, and so on, the actual number of pathnames can be higher.

topology_pathnames

```
EMStatus topology_pathnames(
    const EMTopoNodeDn& parent_dn,
    const RWCString& name,
    RWTValSlistRWCString& pathnames
) const;
```

The type `topology_pathnames` is similar to `view_topology_pathnames()` except that this version works for any type of node. This method will also not send any data requests to the MIS if the *cache_view_graph* optimization is turned on. The reason that the parameters *parent_dn* and *name* are required rather than just the `EMTopoNodeDn` of the node is because these two pieces of information are not cached by the Topology API — the topology view cache has this information but only for view nodes. By having the client application pass this information in, the Topology API can take advantage of the cases where the client already has this information in memory.

find_root_types

```
EMStatus find_root_types(
    RWTValSlistEMTopoTypeDn& types
) const;
```

Returns in *types* a list of all root types, that is. types with no `EMTopoType::base_type`. In the default installation of Solstice EM, the root types are `EMTopoTypeDn::container`, `EMTopoTypeDn::device`, `EMTopoTypeDn::monitor`, and `EMTopoTypeDn::link`.

Note – Unlike the `find_root_nodes()` method, this method will only return the root types on the local MIS.

find_all_types

```
EMStatus find_all_types(
    RWTValSlistEMTopoTypeDn& types
) const;
```

Similar to find_root_types() except that *types* will contain all topology types on the local MIS.

The following methods provide a way to find the agent which is responsible for a particular *managed_object*. If a match is not found, then EMStatus::key_not_found error will result. These methods are used internally by the EMTopoNode class to calculate the EMTopoNode::snmp_agents, EMTopoNode::rpc_agents, and EMTopoNode::cmip_agents attributes from the EMTopoNode::managed_objects attribute.

managed_object_to_cmip_agent

```
EMStatus managed_object_to_cmip_agent(
    const RWCString& managed_object,
    EMCmipAgentDn& cmip_agent
) const;
```

If the object represented by the parameter *managed_object* is contained under a branch of the MIT which is managed by a cmip agent, the unique identifier of the cmip configuration object will be returned in *cmip_agent* and an EMStatus::Success is returned by the function. Otherwise, EMStatus::Key_not_found is returned.

managed_object_to_snmp_agent

```
EMStatus managed_object_to_snmp_agent(
    const RWCString& managed_object,
    EMSnmpAgentDn& snmp_agent
) const;
```

If the object represented by the parameter *managed_object* is contained under a branch of the MIT which is managed by a snmp agent, the unique identifier of the snmp configuration object will be returned in *snmp_agent* and an EMStatus::Success is returned by the function. Otherwise, EMStatus::Key_not_found is returned.

managed_object_to_rpc_agent

```
EMStatus managed_object_to_rpc_agent(
    const RWCString& managed_object,
    EMRpcAgentDn& rpc_agent
) const;
```

If the object represented by the parameter *managed_object* is contained under a branch of the MIT which is managed by a rpc agent, the unique identifier of the rpc configuration object will be returned in *rpc_agent* and an `EMStatus::Success` is returned by the function. Otherwise, `EMStatus::Key_not_found` is returned.

8.13 EMObject Class

Inheritance: none

```
#include topo_api/topo_api.hh
```

The `EMObject` class is an abstract base class that specifies the interface supported by all the persistent object classes (POC): `EMTopoNode`, `EMTopoType`, `EMCmipAgent`, `EMRpcAgent`, and `EMSnpAgent`.

Each POC instance is an interface to a particular set of objects in the MIT. For more information on which MIT objects the five POCs map to, refer to Section 8.2.1, “Relationship to the GDMO.” Each unit of persistent state is called an attribute, and an object is made up of a set of these attributes. Note that each POC attribute can translate to one, several, or no GDMO attribute(s) in the corresponding object(s) in the MIT.

To create a new object in the MIS, first set the mandatory attributes required for creation, must be set either by loading values from another object or setting the values explicitly, using the POC’s access methods.

Either `create_with_all_attributes()`, or `create_with_some_attributes()` is called to create the object in the MIS. Note that `create_with_all_attributes()` only uses attributes that have been given a value. If the create method succeeds, then the `POC::dn` attribute will be set with the unique identifier of the new object.

To destroy an object, first the `POC::dn` identifier must be set, then the `destroy()` method may be called to delete the object from the MIS. This is a permanent, non-reversible operation; some care when using this method.

In order to get the attribute values of a particular object, first set the `POC::dn` identifier, then call either `load_all_attributes()` or `load_some_attributes()`. Once the attribute values are loaded, they stay cached within the POC and remain constant even if the values change in the MIS.

In order to set the attribute values persistently in the MIS, first set the `POC::dn` attribute, then call either `store_all_attributes()` or `store_some_attributes()`. Note that `store_all_attributes()` only stores those attributes that have been given a value.

As a point of reference, the persistence model used is a simplified version of the PMI's Image class.

Enum

```
enum EMObjectOperation {
    em_load,
    em_create,
    em_store,
    em_num_object_operations
};
```

8.13.1 Constructors and Destructor

```
virtual ~EMObject();
```

Because this is an abstract base class, no instances of `EMObject` can be created.

8.13.2 EMObject Member Functions Supported By POC Classes

`exists`

```
virtual RWBoolean exists() const;
```

Returns `TRUE` if the object represented by the persistent object class instance exists in the MIT.

Note – You must have the unique identifier (`EMTopoNode::dn`, `EMTopoType::dn`, etc.) set for this method to work properly. Otherwise, `FALSE` will be returned.

The following two methods create a new object in the MIS. In order for the create to succeed, the mandatory attributes required by the particular POC must be set. The new object has its attribute values determined as follows:

- If `create_with_all_attributes()` was used, then any attribute that was given a value will be stored in the new object.
- If the function `create_with_some_attributes()` is used, then only the specified attributes are stored in the new object.

In either case, any attributes that are not given a value will take on a default value defined by the GDMO for that object.

The possible error conditions are `EMStatus::missing_mandatory_attribute`, `EMStatus::attribute_is_not_creatable`, `EMStatus::encode_error`, and `EMStatus::pmi_error`.

`create_with_all_attributes`

```
virtual EMStatus create_with_all_attributes();
```

`create_with_some_attributes`

```
virtual EMStatus create_with_some_attributes(  
    const EMIntegerSet& attributes  
);
```

`destroy`

```
virtual EMStatus destroy();
```

This method deletes the object identified by `POC::dn` from the MIS. This is a permanent, non-reversible operation; some care should be taken when using this method.

The possible error conditions are `EMStatus::missing_mandatory_attribute`, `EMStatus::object_doesnt_exist`, and `EMStatus::pmi_error`.

`load_all_attributes`

```
virtual EMStatus load_all_attributes();  
virtual EMStatus load_some_attributes(  
    const EMIntegerSet& attributes  
);
```

This method loads attributes of the object identified by `POC::dn` from the MIS into the POC internal cache. `load_all_attributes()` loads all attributes whereas `load_some_attributes()` only loads the specified attributes.

The possible error conditions are `EMStatus::missing_mandatory_attribute`, `EMStatus::object_doesnt_exist`, `EMStatus::not_supported`, and `EMStatus::pmi_error`.

The following methods will store attributes of the object identified by `POC::dn` to the MIS from the POC internal cache.

store_all_attributes

```
virtual EMStatus store_all_attributes();
```

`store_all_attributes()` stores all attributes that have been given a value.

store_some_attributes

```
virtual EMStatus store_some_attributes(  
    const EMIntegerSet& attributes );
```

`store_some_attributes()` stores only the specified attributes, without regard to whether the attributes have been given a value. Care must be taken only to specify attributes that have values; otherwise an arbitrary (usually NULL or empty) value is stored.

The possible error conditions are `EMStatus::missing_mandatory_attribute`, `EMStatus::object_doesnt_exist`, `EMStatus::attribute_is_not_storable`, `EMStatus::encode_error` and `EMStatus::pmi_error`.

get_active_attributes

```
virtual void get_active_attributes(  
    EMIntegerSet& set  
) const;
```

This method returns the set of attributes that have been given a value characteristic of *set*.

The following two methods clear the internal memory of all or some attributes. This is useful when you want to reuse a POC instance to access a different object and do not want the previous values to remain in effect.

After `clear_all_attributes()` is called, all attributes no longer have a value in the internal memory, including the `POC::dn` attribute.

clear_all_attributes

```
virtual void clear_all_attributes();
```

clear_some_attributes

```
virtual void clear_some_attributes(  
    const EMIntegerSet& set  
);
```

8.13.3 Operators Supported by all POC classes

```
RWBoolean operator ==(  
    const EMPOC&  
);
```

Two instances of EMPOC are considered to be equal if they each have the same attributes with a value and those values are the same for both. If one instance has a value for an attribute for which the other instance does not have a value, then the instances are not equal to one another. To compare a subset of the attributes, use `compare_some_attributes()`.

And the *not*-operator,

```
RWBoolean operator !=(  
    const EMPOC&  
);
```

8.13.4 Other Member Functions Supported by POC Classes.

`compare_all_attributes`

```
RWBoolean compare_all_attributes(  
    const EMPOC& other_poc  
    ) const;
```

The method `compare_all_attributes()` is equivalent to `operator== ()`.

`compare_some_attributes`

```
RWBoolean compare_some_attributes(  
    const EMPOC& other_poc,  
    const EMIntegerSet& attributes  
    ) const;
```

The method `compare_some_attributes()` compares only the specified subset of attributes. For each attribute in *attributes*, either both EMPOC instances must have no value set for the attribute or if both of them have a value set for the attribute then the values must be equal. If one EMPOC instance has a value for the attribute while the other does not, then the instances are not equal.

The following functions are used to compare set attributes and are equivalent to the various set operations provided by other functions.

`diff_all_attributes`

```

RWBoolean diff_all_attributes(
    const EMPOC& other_agent,
    EMIntegerSet& differences
) const;

```

`diff_some_attributes`

```

RWBoolean diff_some_attributes(
    const EMPOC& other_agent,
    const EMIntegerSet& attributes,
    EMIntegerSet& differences
) const;

```

The `diff_all_attributes()`, and `diff_some_attributes()` methods both have a return value equal to `compare_all_attributes()` and `compare_some_attributes()`, respectively, and in addition, return the set of attributes where the two instances differed, if any. The class `EMIntegerSetIterator` can then be used to iterate over the *differences*.

8.13.5 Static Member Functions Supported by POC Classes

These static methods provide information about the EMPOC's attributes.

`get_valid_attributes`

```

static const EMIntegerSet& get_valid_attributes(
    EMObjectOperation op = em_load
);

```

The method `get_valid_attributes()` returns the set of attributes that are valid for the specified operation *op*.

get_mandatory_attributes

```
static const EMIntegerSet& get_mandatory_attributes(
    EMObjectOperation op = em_load
);
```

The method `get_mandatory_attributes()` returns the set of attributes that are mandatory for the specified operation *op*. If a mandatory attribute is not set when the particular operation is called, an `EMStatus::missing_mandatory_attribute` error will result.

get_attribute_name

```
static const RWCString& get_attribute_name(
    EMPOC::Attribute attribute
);
```

This method returns an attribute name in string form.

8.14 EMTopoNodeDn Class

Inheritance: none

```
#include topo_api/topo_api.hh
```

An instance of the `EMTopoNodeDn` class uniquely identifies one topology node out of the set of topology node objects interfaced by the `EMTopoNode` class.

Enum

```
enum NullId {  
    null_id = -1  
};
```

8.14.1 Constructors and Destructor

The following are constructors for EMTopoNodeDn:

```
EMTopoNodeDn( );
```

The default constructor creates a EMTopoNodeDn instance that is null. The `is_null()` method returns TRUE for this instance.

```
EMTopoNodeDn(  
    const RWCString& system_name,  
    long unique_id  
);
```

Creates an EMTopoNodeDn instance that is uniquely identified by *system_name* and *unique_id*.

And the default destructor,

```
~EMTopoNodeDn( );
```

8.14.2 Operators

```
RWBoolean operator ==(
const EMTopoNodeDn& dn
) const;
```

Two instances are equal if they have both the same system name and the same unique name or if they are both null.

And the *not*-operator,

```
RWBoolean operator !=(
const EMTopoNodeDn& dn
) const;
```

8.14.3 Access Member Functions

The following are access methods for the EMTopoNodeDn class.

`system_name`

```
const RWCString& system_name() const;

void system_name(
    const RWCString& system_name
);
```

This method returns the name of the MIS where the topology node is stored.

unique_id

```
long unique_id() const;

void unique_id(
    long unique_id
);
```

The topology node identifier is unique within a single MIS.

8.14.4 General Member Functions

make_null

```
void make_null();
```

Set to null value. A null value means that the EMTopoNodeDn does not refer to any topology node.

is_null

```
RWBoolean is_null() const;
```

Test for null value. A null value means that the EMTopoNodeDn does not refer to any topology node.

8.14.5 Related Global Operators

```
ostream& operator<<(ostream& s, const EMTopoNodeDn& dn);
```

The stream output operator << is defined to provide an easy way to print out the value of EMTopoNodeDn.

8.15 EMTopoNode Class

Inheritance: EMObject

```
#include topo_api/topo_api.hh
```

The EMTopoNode class represents a topology node, which is the unit of management in Solstice EM. Using the standard POC methods, you can create, delete, and compare topology nodes. Using the EMTopoNode's access methods you can get and set the name, topology pathname, logical and geographical location, topology type, and associated managed objects and their corresponding CMIP, RPC, and/or SNMP agent objects among other attributes. The EMTopoNode class also provides a callback mechanism to notify clients when a topology node has been created, deleted, or has had one or more attributes changed.

In TABLE 8-4, the attribute key is:

- C – Attribute can be set at creation time.
- S – Attribute can be set after creation time.
- M – Mandatory; attribute must be set for operation to succeed.
- X – Allowed; attribute can be set as an option.

TABLE 8-4 EMTopoNode Attributes Table

Attribute Enum	C	S	Description
dn		M	Unique identifier.
name	M	X	Name of this node (need not be unique).
topology_pathnames			List of all topology pathnames for the node. At a minimum, there will be one pathname for each parent. However, since each parent can also have more than one parent, and so on, the actual number of pathnames may be higher. Example list: “/Root/Internet/129.146.74.0/host-45”, “/Root/hosts/host-45”.
type_name	M	X	Type name of this node.
managed_objects	X	X	List of DNs (in ASCII slash format) of the managed objects in the MIT associated with this node.
cmip_agents	X	X	List of CMIP agents which have managed objects listed as part of managed_objects attribute.

TABLE 8-4 EMTopoNode Attributes Table (*Continued*)

Attribute Enum	C	S	Description
snmp_agents	X	X	List of SNMP agents which have managed objects listed as part of managed_objects attribute.
rpc_agents	X	X	List of RPC agents which have managed objects listed as part of managed_objects attribute.
parents	M	X	List of topology nodes that contain this node in the topology directed acyclic graph (DAG).
children			List of topology nodes that are contained by this node.
view_children			Subset of children whose type_name is a view; that is, EMTopoType::is_view(type_name) returns TRUE.
links	X	X	List of Link topology nodes connected to this node.
propagate_peers	X	X	List of topology nodes for this node's severity propagation if is_severity_attribute is TRUE.
is_severity_propagated	X	X	If TRUE, then the node's propagated_severity will be propagated to each of the nodes in its parent and propagated peers hierarchy, where it will factor in the calculation of their propagated_severity.
state	X	X	Can be used to store an integer value.
severity			If the Alarm Service is running, the severity indicates the highest severity alarm posted against any of the managed_objects. Note: Normally, an application never sets the severity attribute; this attribute is automatically updated from the Alarm Service.
propagated_severity			If the Alarm Service is running, the propagated_severity indicates the highest severity among this node's severity and the propagated_severity of any children of this node who have their is_severity_propagated flag set to TRUE.
display_statuses	X	X	A user-defined list of tags, value pairs, such as { { "CPUUsage", 45 }, { "DiskLoad" , 2345 } }.
geographical_location	X	X	The latitude and longitude in degrees floating-point of the location of this node.
layer_name	X	X	The layer to which this node belongs.

TABLE 8-4 EMTopoNode Attributes Table (*Continued*)

Attribute Enum	C	S	Description
user_data	X	X	User-defined data that should contain values for each attribute name listed in <code>EMTopoType::user_data_attribute_names</code> for the <i>type_name</i> of this node.
logical_locations	X	X	A list of locations where the node appears in each of its parent views.
view_background_image_filename	X	X	Absolute pathname of Sun raster file image to be displayed when the viewer canvas is in logical view mode.
view_map_config_filename	X	X	Absolute pathname of geographical map configuration (GMC) file to be displayed when the Viewer canvas is in geographical view mode.
view_default_geo_area	X	X	Default geographical area (specified as a center and view width in km) to be displayed when the <code>view_map_config_filename</code> is first displayed.
monitor_rotation	X	X	Number of degrees to rotate the monitor node.
monitor_visible_children		X	Subset of the children list of nodes that should appear in the monitor sections.
monitor_hidden_children			List of children nodes remaining when <code>monitor_visible_children</code> list of nodes are subtracted from the children list of nodes. These nodes do not appear in a monitor section, even if there are empty sections.
monitor_max_visible_children			Maximum number of visible children supported by the particular type of monitor.
array_visible_children		X	List of topology nodes which are displayed in an array. This list is a subset of the preexisting children attribute supported by all containers.
array_hidden_children			List of topology nodes which are not displayed in an array. This list is a subset of the preexisting children attribute supported by all containers.
array_orientation	X	X	The orientation of an array, either horizontal or vertical. This indicates whether the topology nodes grouped by the array should be laid down row by row or column by column.

TABLE 8-4 EMTopoNode Attributes Table (*Continued*)

Attribute Enum	C	S	Description
array_num_columns	X	X	The number of columns in an array with horizontal orientation, or the number of rows in an array with vertical orientation.
array_cell_width	X	X	The width (in characters) of each cell in an array. If the value is 0, then the cell width will be set to the minimum width necessary to display the label of the widest cell.
bus_logical_locations	X	X	List of x,y points which define the bus's shape. Note that the points are constrained so that all line segments are alternatingly horizontal or vertical.

8.15.1 Example

```
#include topo_api/topo_api.hh

// this assumes the snmp_agent was created
// elsewhere
RWBoolean
create_host(
    const EMTopoNodeDn& parent_dn,
    const EMSnmpAgentDn& snmp_agent_dn
)
{
    EMTopoNode node;

    node.set_name(snmp_agent_dn.unique_name());
    node.set_type_name(EMTopoTypeDn::host);
    node.add_parent(parent_dn);
    // for hosts, we use the DN of the snmp agent
    // object as the managed object.
    node.add_managed_object(snmp_agent_dn.slash_form());

    EMStatus status;
    if (!(status = node.create_with_all_attributes())) {
        cerr << "Error: " << status << endl;
        return FALSE;
    }
    return TRUE;
}
```

Enum

```
enum EMTopoNode::AttributeType {  
    all_attributes,  
    common_attributes,  
    view_only_attributes,  
    monitor_only_attributes,  
    link_only_attributes,  
    device_only_attributes,  
    array_only_attributes,  
    bus_only_attributes,  
    num_attribute_types  
};
```

```
enum EMTopoNode::Attribute {  
    dn=0,  
    name,  
    type_name,  
    managed_objects,  
    cmip_agent,  
    snmp_agent,  
    rpc_agent,  
    parents,  
    children,  
    children_containers_only,  
    links,  
    propagate_peers,  
    is_severity_propagated,  
    state,  
    severity,  
    propagated_severity,  
    display_statuses,  
    geographical_location,  
    layer_name,  
    user_data,  
    logical_locations,  
    view_background_image_filename,  
    view_map_config_filename,  
    view_default_geo_area,  
    monitor_rotation,  
    monitor_visible_children,  
    monitor_hidden_children,  
    monitor_max_visible_children,  
    array_visible_children,  
    array_hidden_children,  
    array_orientation,  
    array_num_columns,  
    array_cell_width,  
    bus_logical_locations,  
    num_attributes  
};
```

```
enum EMTopoNode::Severity {
    indeterminate = 0,
    critical = 1,
    major = 2,
    minor = 3,
    warning = 4,
    cleared = 5,
    min_severity = indeterminate,
    max_severity = cleared,
    num_severity = max_severity - min_severity + 1
};
```

Structs

```
struct EMTopoNode::Location
{
    long x;
    long y;
    long z;

    Location();
    Location(long p_x, long p_y, long p_z=0);
    RWBoolean operator == (const Location& l) const;
    RWBoolean operator != (const Location& l) const;

    ostream& operator<<(ostream& s, const Location& l);
};
```

```

struct EMTopoNode::LocationInParent
{
    EMTopoNodeDn parent;
    Location location;

    LocationInParent();

    LocationInParent(
        const EMTopoNodeDn& p_parent,
        const Location& p_location
    );

    RWBoolean operator == (
        const LocationInParent& l
    ) const;
    RWBoolean operator != (
        const LocationInParent& l
    ) const;

    ostream& operator<<(
        ostream& s,
        const LocationInParent& l
    );
};

```

```

struct EMTopoNode::GeoLocation
{
    double longitude;
    double latitude;

    GeoLocation();
    GeoLocation(double p_longitude, double p_latitude);
    RWBoolean operator == (const GeoLocation& l) const;
    RWBoolean operator != (const GeoLocation& l) const;

    ostream& operator<<(ostream& s, const GeoLocation& l);
};

```



```

struct EMTopoNode::DisplayStatus
{
    RWCString label;
    long value;

    DisplayStatus();
    DisplayStatus(const RWCString& p_label, long p_value);
    RWBoolean operator == (const DisplayStatus& d) const;
    RWBoolean operator != (const DisplayStatus& d) const;

    ostream& operator<<(ostream& s, const DisplayStatus& d);
};

```

```

struct EMTopoNode::UserDatum
{
    RWCString attribute_name;
    Morf value;

    UserDatum();
    UserDatum(
        const RWCString& p_attribute_name,
        const Morf& p_value);
    RWBoolean operator == (
        const UserDatum& d
    ) const;
    RWBoolean operator != (
        const UserDatum& d) const;
    ostream& operator<<(ostream& s,
        const UserDatum& d);
};

```

8.15.2 Constructors and Destructor

```
EMTopoNode ( ) ;
```

```
EMTopoNode (
    const EMTopoNodeDn& dn
) ;
```

```
EMTopoNode (
    const EMTopoNode& node
) ;
```

The default destructor,

```
~EMTopoNode ( ) ;
```

8.15.3 Access Member Functions

The member function methods of the EMTopoNode class refer to the attributes listed in the table at the beginning of this section, table 6-4. See the table for a description of each of the attributes.

get_dn

```
EMStatus get_dn (
    EMTopoNodeDn& dn
) const ;
```

This function gets the distinguished name, *dn*, associated with a node.

set_dn

```
EMStatus set_dn(  
    const EMTopoNodeDn& dn  
);
```

This function sets the distinguished name, *dn*, associated with a node.

get_name

```
EMStatus get_name(  
    RWCString& name  
) const;
```

This function gets the name of a node.

set_name

```
EMStatus set_name(  
    const RWCString& name  
);
```

This function sets the name of a node.

get_topology_pathnames

```
EMStatus get_topology_pathnames(  
    RWTValSlistRWCString& topology_pathnames  
) const;
```

This function gets a list of all the pathnames for a specific node.

get_type_name

```
EMStatus get_type_name(  
    RWCString& type_name  
) const;
```

This function gets the type name of a specific node.

set_type_name

```
EMStatus set_type_name(  
    const RWCString& type_name  
);
```

This function sets the type name of a specific node.

get_severity

```
EMStatus get_severity(  
    EMTopoNode::Severity& severity  
) const;
```

This function gets the severity associated with a node.

set_severity

```
EMStatus set_severity(  
    EMTopoNode::Severity severity  
);
```

This function sets the severity associated with a node, but now has no effect. Only the Alarm Service will update this attribute.

get_propagated_severity

```
EMStatus get_propagated_severity(
    EMTopoNode::Severity& severity
) const;
```

This function gets the propagated severity associated with a node.

get_is_severity_propagated

```
EMStatus get_is_severity_propagated(
    RWBoolean& is_severity_propagated
) const;
```

This function gets a node's *is_severity_propagated* and determines whether the property of *propagated_severity* should be propagated to each of the parent and peer nodes, where it is a factor in the calculation of their severity.

set_is_severity_propagated

```
EMStatus set_is_severity_propagated(
    const RWBoolean& is_severity_propagated
);
```

This function sets a node's *is_severity_propagated* and determines whether the property of *propagated_severity* should be propagated to each of the parent and peer nodes, where it is a factor in the calculation of their severity.

get_propagate_peers

```
EMStatus get_propagate_peers(
    RWTValSlistEMTopoNodeDn& peers
) const;
```

This function gets a list of *propagate_peer* nodes in the topology.

set_propagate_peers

```
EMStatus set_propagate_peers(  
    const RWTValSlistEMTopoNodeDn& peers  
);
```

This function sets a list of `propagate_peer` nodes in the topology.

add_propagate_peer

```
EMStatus add_propagate_peer(  
    const EMTopoNodeDn& peer  
);
```

This function adds a topology node to the `propagate_peers` list.

remove_propagate_peer

```
EMStatus remove_propagate_peer(  
    const EMTopoNodeDn& peer  
);
```

This function removes a topology node from the `propagate_peers` list.

get_state

```
EMStatus get_state(  
    long& state  
) const;
```

This function is used to get the state associated with a node.

`set_state`

```
EMStatus set_state(
    long state
);
```

This function is used to set the state associated with a node.

`get_display_statuses`

```
EMStatus get_display_statuses(
    RWTValSlistEMTopoNode::DisplayStatus& statuses
) const;
```

This function gets the user-defined list of tag-value pairs for a number of display statuses.

`set_display_statuses`

```
EMStatus set_display_statuses(
    const RWTValSlistEMTopoNode::DisplayStatus& statuses
);
```

This function sets the user-defined list of tag-value pairs for a number of display statuses.

`get_display_status`

```
EMStatus get_display_status(
    const RWCString& tag,
    long& value
) const;
```

This function will get a *value* for the specified *tag*, if it exists. If not, `EMStatus:key_not_found` will be returned.

add_display_status

```
EMStatus add_display_status(  
    const RWCString& tag,  
    long value  
);
```

This function will add a tag-value pair to the `display_status` attribute. If the *tag* already exists, the value will be replaced by *value*.

remove_display_status

```
EMStatus remove_display_status(  
    const RWCString& tag  
);
```

This function will remove the specified tag-value pair from the `display_status` attribute.

get_children

```
EMStatus get_children(  
    RWTValSlistEMTopoNodeDn& children  
) const;
```

This function will get a list of topology nodes that are contained by this node.

get_children_containers_only

```
EMStatus get_children_containers_only(  
    RWTValSlistEMTopoNodeDn& children  
) const;
```

This function gets the subset of topology nodes contained by this node which are views, that is, the function `EMTopoPlatform::instance()->is_view` returns `TRUE` for each of the applicable topology nodes.

get_parents

```
EMStatus get_parents(  
    RWTValSlistEMTopoNodeDn& parents  
) const;
```

This function gets a list of topology nodes that are parents.

set_parents

```
EMStatus set_parents(  
    const RWTValSlistEMTopoNodeDn& parents  
);
```

This function sets the parents of a particular node.

add_parent

```
EMStatus add_parent(  
    const EMTopoNodeDn& parent  
);
```

This function adds a node parent to the parent's list.

remove_parent

```
EMStatus remove_parent(  
    const EMTopoNodeDn& parent  
);
```

This function removes the node parent from the parents list.

get_links

```
EMStatus get_links(  
    RWTValSlistEMTopoNodeDn& links  
) const;
```

This function gets a list of links to topology nodes connected to this node.

set_links

```
EMStatus set_links(  
    const RWTValSlistEMTopoNodeDn& links  
);
```

This function sets a list of links to topology nodes connected to this node.

add_link

```
EMStatus add_link(  
    const EMTopoNodeDn& link  
);
```

This function adds a topology node link to nodes connected to this node.

remove_link

```
EMStatus remove_link(  
    const EMTopoNodeDn& link  
);
```

This function removes a topology node link from the list of nodes connected to this node.

get_logical_location

```
EMStatus get_logical_location(  
    const EMTopoNodeDn& parent,  
    EMTopoNode::Location& location  
) const;
```

This function gets the logical location of the node in its parent view *parent*. If the node is not contained in *parent*, the `EMStatus::key_not_found` status string is returned.

set_logical_location

```
EMStatus set_logical_location(  
    const EMTopoNodeDn& parent,  
    const EMTopoNode::Location& location  
);
```

This function sets the logical location of the node in its parent view *parent*. If the node is not contained in *parent*, the `EMStatus::key_not_found` status string is returned.

get_logical_locations

```
EMStatus get_logical_locations(  
    RWTValSlistEMTopoNode::LocationInParent& locations  
) const;
```

This function gets a list of logical locations where the node appears in each of its parent views.

set_logical_locations

```
EMStatus set_logical_locations(
    const RWTValSlistEMTopoNode::LocationInParent& locations
);
```

This function sets a list of logical locations where the node appears in each of its parent views. The node must already be contained within each parent specified in `EMTopoNode::LocationInParent` struct.

get_geographical_location

```
EMStatus get_geographical_location(
    EMTopoNode::GeoLocation& location,
    RWBoolean& is_null
) const;
```

This function gets the latitude and longitude in degrees floating-point of the location of this node. If *is_null* is `TRUE`, then the node has no geographical position, and *location* is undefined.

set_geographical_location

```
EMStatus set_geographical_location(
    const EMTopoNode::GeoLocation& location,
    RWBoolean is_null = FALSE
);
```

This function sets the latitude and longitude in degrees floating-point of the location of this node. Passing *is_null* as `TRUE`, will result in the node having no geographical position, and *location* is undefined.

get_layer_name

```
EMStatus get_layer_name(  
    RWCString& name  
) const;
```

This function gets the layer that a specific node belongs to.

set_layer_name

```
EMStatus set_layer_name(  
    const RWCString& name  
);
```

This function sets the layer that a specific node belongs to.

get_managed_objects

```
EMStatus get_managed_objects(  
    RWTValSlistRWCString& managed_objects  
) const;
```

This function gets a list of DNs (in ASCII slash format) of the managed objects in the MIT associated with this node.

set_managed_objects

```
EMStatus set_managed_objects(  
    const RWTValSlistRWCString& managed_objects  
);
```

This function sets a list of DNs (in ASCII slash format) of the managed objects in the MIT associated with this node.

add_managed_object

```
EMStatus add_managed_object(  
    const RWCString& managed_object  
);
```

This function adds a DN (in ASCII slash format) to the list of managed objects in the MIT associated with this node.

remove_managed_object

```
EMStatus remove_managed_object(  
    const RWCString& managed_object  
);
```

This function removes a DN (in ASCII slash format) from the list of managed objects in the MIT associated with this node. If *managed_object* is not in the list, then `EMStatus::does_not_exist` will be returned.

get_cmip_agents

```
EMStatus get_cmip_agents(  
    RWTValSlistEMCmipAgentDn& agents  
    ) const;
```

This function gets a list of CMIP agents which have managed objects listed as part of `managed_objects` attribute.

get_snmp_agents

```
EMStatus get_snmp_agents(  
    RWTValSlistEMSnmpAgentDn& agents  
    ) const;
```

This function gets a list of SNMP agents which have managed objects listed as part of `managed_objects` attribute.

get_rpc_agents

```
EMStatus get_rpc_agents(  
    RWTValSlistEMRpcAgentDn& agents  
    ) const;
```

This function gets a list of RPC agents which have managed objects listed as part of `managed_objects` attribute.

get_user_data

```
EMStatus get_user_data(  
    RWTValSlistEMTopoNode::UserDatum& user_data  
    ) const;
```

This function gets user-defined data that should contain values for each attribute name listed in `EMTopoType::user_data_attribute_names` for the *type_name* of this node.

set_user_data

```
EMStatus set_user_data(  
    const RWTValSlistEMTopoNode::UserDatum& user_data  
    );
```

This function sets user-defined data that should contain values for each attribute name listed in `EMTopoType::user_data_attribute_names` for the *type_name* of this node.

get_user_datum

```
EMStatus get_user_datum(  
    const RWCString& attribute_name,  
    Morf& morf  
    ) const;
```

This function gets a user-defined datum corresponding to one of the attribute names listed in `EMTopoType::user_data_attribute_names` for the *type_name* of this node.

add_user_datum

```
EMStatus add_user_datum(  
    const RWCString& attribute_name, const Morf& morf  
    );
```

This function adds a user-defined datum for one of the attribute names listed in `EMTopoType::user_data_attribute_names` for the *type_name* of this node. If *attrib_name* is not valid, then `EMStatus::Key_not_found` is returned.

remove_user_datum

```
EMStatus remove_user_datum(  
    const RWCString& attribute_name  
    );
```

This function removes a user-defined datum for one of the attribute names listed in `EMTopoType::user_data_attribute_names` for the *type_name* of this node. If *attrib_name* is not valid, then `EMStatus::Key_not_found` is returned.

get_view_background_image_filename

```
EMStatus get_view_background_image_filename(  
    RWCString& filename  
    ) const;
```

This function gets the absolute pathname of the Sun raster file image to be displayed when the viewer canvas is in logical view mode.

set_view_background_image_filename

```
EMStatus set_view_background_image_filename(  
    const RWCString& filename  
    );
```

This function set the absolute pathname of the Sun raster file image to be displayed when the viewer canvas is in logical view mode.

get_view_map_config_filename

```
EMStatus get_view_map_config_filename(  
    RWCString& filename  
    ) const;
```

This function gets the absolute pathname of the geographical map configuration (GMC) file to be displayed when the Viewer canvas is in geographical view mode.

set_view_map_config_filename

```
EMStatus set_view_map_config_filename(  
    const RWCString& filename  
    );
```

This function sets the absolute pathname of the geographical map configuration (GMC) file to be displayed when the Viewer canvas is in geographical view mode.

get_view_default_geo_area

```
EMStatus get_view_default_geo_area(  
    EMTopoNode::GeoLocation& center,  
    double& width_in_km,  
    RWBoolean& is_null  
    ) const;
```

This function gets the default geographical area (specified as a center and view width in km) to be displayed when the `view_map_config_filename` is first displayed.

set_view_default_geo_area

```
EMStatus set_view_default_geo_area(  
    const EMTopoNode::GeoLocation& center,  
    double width_in_km,  
    RWBoolean is_null = FALSE  
    );
```

This function sets the default geographical area (specified as a center and view width in km) to be displayed when the `view_map_config_filename` is first displayed.

get_monitor_rotation

```
EMStatus get_monitor_rotation(  
    long& rotation,  
    RWBoolean& is_null  
    ) const;
```

This function gets the number of degrees to rotate the monitor node.

`set_monitor_rotation`

```
EMStatus set_monitor_rotation(
    long rotation,
    RWBoolean is_null = FALSE
);
```

This function sets the number of degrees to rotate the monitor node.

`get_monitor_visible_children`

```
EMStatus get_monitor_visible_children(
    RWTValSlistEMTopoNodeDn& children
) const;
```

This function gets the subset of the children list of nodes that should appear in the monitor sections.

`set_monitor_visible_children`

```
EMStatus set_monitor_visible_children(
    const RWTValSlistEMTopoNodeDn& children
);
```

This function sets the subset of the children list of nodes that should appear in the monitor sections.

`add_monitor_visible_child`

```
EMStatus add_monitor_visible_child(
    const EMTopoNodeDn& child
);
```

This function adds *child* to the subset of the children list of nodes that should appear in the monitor sections.

remove_monitor_visible_child

```
EMStatus remove_monitor_visible_child(  
    const EMTopoNodeDn& child  
);
```

This function removes *child* from the subset of the children list of nodes that should appear in the monitor sections.

get_monitor_hidden_children

```
EMStatus get_monitor_hidden_children(  
    RWTValSlistEMTopoNodeDn& children  
    ) const;
```

This function gets the list of children nodes remaining when *monitor_visible_children* list of nodes are subtracted from the children list of nodes. These nodes do not appear in a monitor section, even if there are empty sections.

get_monitor_max_visible_children

```
EMStatus get_monitor_max_visible_children(  
    long& max_children  
    ) const;
```

This function gets the maximum number of visible children supported by the particular type of monitor.

get_array_orientation

```
EMStatus get_array_orientation(  
    ArrayOrientation& orientation  
    ) const;
```

The *get_array_orientation* function returns the *array_orientation* attribute in *orientation*. It returns `EMStatus::attribute_not_set` if the *array_orientation* attribute has not been initialized.

set_array_orientation

```
EMStatus set_array_orientation(  
    ArrayOrientation orientation  
);
```

The `set_array_orientation` function sets the `array_orientation` attribute to *orientation*.

get_array_num_columns

```
EMStatus get_array_num_columns(  
    unsigned long& num_columns  
) const;
```

The `get_array_num_columns` function returns the `array_num_columns` attribute in *num_columns*. It returns `EMStatus::attribute_not_set` if the `array_num_columns` attribute has not been initialized.

set_array_num_columns

```
EMStatus set_array_num_columns(  
    unsigned long num_columns  
);
```

The `set_array_num_columns` function sets the `array_num_columns` attribute to *num_columns*. If *num_columns* equals `array_num_columns_autosize`, then the `num_columns` is dynamically selected in order to make the array cells layout as close as possible to a square.

get_array_cell_width

```
EMStatus get_array_cell_width(  
    unsigned long& cell_width  
    ) const;
```

The `get_array_cell_width` function returns the `array_cell_width` attribute in *cell_width*. It returns `EMStatus::attribute_not_set` if the `array_cell_width` attribute has not been initialized.

set_array_cell_width

```
EMStatus set_array_cell_width(  
    unsigned long cell_width  
    );
```

The `set_array_cell_width` function sets the `array_cell_width` attribute to *cell_width*. If *cell_width* is equal to `array_cell_width_autosize`, then the `cell_width` is dynamically adjusted to fit the cell of the cell with the longest label.

get_array_visible_children

```
EMStatus get_array_visible_children(  
    RWTValSlistEMTopoNodeDn& children  
    ) const;
```

The `get_array_visible_children` function returns the `array_visible_children` attribute in *children*. It returns `EMStatus::attribute_not_set` if the `array_visible_children` attribute has not been initialized.

set_array_visible_children

```
EMStatus set_array_visible_children(  
    const RWTValSlistEMTopoNodeDn& children  
);
```

The `set_array_visible_children` function sets the `array_visible_children` attribute to *children*.

add_array_visible_child

```
EMStatus add_array_visible_child(  
    const EMTopoNodeDn& child  
);
```

The `add_array_visible_child` function adds a node, *child*, to the `array_visible_children` attribute. It returns `EMStatus::attribute_not_set` if the `array_visible_children` attribute has not been initialized, and `EMStatus::already_exists` if *child* is already a member of `array_visible_children`.

remove_array_visible_child

```
EMStatus remove_array_visible_child(  
    const EMTopoNodeDn& child  
);
```

The `remove_array_visible_child` function removes a node, *child*, from the `array_visible_children` attribute. It returns `EMStatus::attribute_not_set` if the `array_visible_children` attribute has not been initialized, and `EMStatus::does_not_exist` if *child* is not a member of `array_visible_children`.

get_array_hidden_children

```
EMStatus get_array_hidden_children(  
    RWTValSlistEMTopoNodeDn& children  
    ) const;
```

The `get_array_hidden_children` function returns the `array_hidden_children` attribute in *children*. It returns `EMStatus::attribute_not_set` if the `array_hidden_children` attribute has not been initialized.

get_bus_logical_locations

```
EMStatus get_bus_logical_locations(  
    RWTValSlistLocation& logical_locations  
    ) const;
```

The `get_bus_logical_locations` function returns the `bus_logical_locations` attribute in *logical_locations*. It returns `EMStatus::attribute_not_set` if the `array_visible_children` attribute has not been initialized.

set_bus_logical_locations

```
EMStatus get_bus_logical_locations(  
    const RWTValSlistLocation& logical_locations  
    );
```

The `set_bus_logical_locations` function sets the `bus_logical_locations` attribute to *logical_locations*.

8.15.4 Static Member Functions for Event Subscription

The `EMTopoNode` class provides an event subscription service so that clients can be notified when a topology node is created, deleted, or modified.

```
struct EMTopoNodeCallbackData
{
    EMEventType event_type;
    EMTopoNodeDn node_dn;
    EMTopoNode changes;
    void* client_data;
};

typedef void (*Callback)(
    const EMTopoNodeCallbackData& cbd
);
```

To register for events, the client must provide a callback function with of type `EMTopoNode::Callback`. When the client's callback is called, the `cbd` parameter will be filled in with information about the event. The `event_type` field indicates the type of event: `em_create_event`, `em_delete_event`, or `em_change_event`. The `node_dn` parameter uniquely identifies the topology node that was created, deleted, or modified. For `em_change_event` only, the `changes` parameter will contain the new values for all attributes which changed. To get a list of the changed attributes, call `EMTopoNode::get_active_attributes()`. The normal `EMTopoNode` access methods may be used to get the new attribute values. Finally, the `client_data` field is the same as the `client_data` parameter of `EMTopoNode::register_callback()`. An `EMTopoNode` events example is `$EM_HOME/src/topo_api/topo_events.cc`.

```
void register_callback(
    EMEventType event,
    Callback callback,
    void* client_data
);
```

Registers *callback* to be called when *event* occurs on any topology node. If *event* equals `em_any_event`, then *callback* will be called for any of `em_create_event`, `em_delete_event`, or `em_change_event`. The parameter *client_data* will be used to initiaze the *client_data* field in the `EMTopoNodeCallbackData` struct.

Note – If the same *callback* has already been registered for the same *event*, then the callback will not be added a second time. However, the *client_data* will replace the previous *client_data*.

```
static void unregister_callback(  
    EMTopoNode::Callback callback,  
    void* user_callback_data  
);
```

Removes *callback* that was previously registered.

8.15.5 Related Global Operators

```
ostream& operator<<(  
    ostream& s,  
    const EMTopoNode& node  
);
```

The stream output operator << is defined to provide an easy way to print out the attribute values of EMTopoNode.

```
ostream& operator<<(  
    ostream& s,  
    const EMTopoTypeDn& dn  
);
```

And the Assignment Operator,

```
EMTopoNode& operator =(  
    const EMTopoNode&  
);
```

8.16 EMTopoTypeDn Class

Inheritance: none

```
#include topo_api/topo_api.hh
```

An instance of the EMTopoTypeDn class uniquely identifies one topology type out of the set of topology types interfaced by the EMTopoType.

8.16.1 Constants

```
static const RWCString  
    EMTopoTypeDn::container,  
    EMTopoTypeDn::device,  
    EMTopoTypeDn::link,  
    EMTopoTypeDn::monitor,  
    EMTopoTypeDn::array,  
    EMTopoTypeDn::bus,  
    EMTopoTypeDn::sun;
```

Convenience constants for the default base types.

8.16.2 Constructors and Destructor

The default constructor creates a null object.

```
EMTopoTypeDn();
```

The following constructor takes the MIS name where the object is stored, and the topology type name.

```
EMTopoTypeDn(  
    const RWCString& system_name,  
    const RWCString& unique_name  
);
```

The default destructor,.

```
~EMTopoTypeDn();
```

8.16.3 Operators

The following is the logical equivalence operator:

```
RWBoolean operator ==(  
    const EMTopoTypeDn& dn  
) const;
```

Two instances are equal if they have both the same system name and the same unique name or if they are both null.

And the *not*-operator,

```
RWBoolean operator !=(  
    const EMTopoTypeDn& dn  
) const;
```

8.16.4 Access Member Functions

The following are access methods for the EMTopoTypeDn class.

system_name

```
const RWCString& system_name() const;

void system_name(
    const RWCString& system_name
);
```

The name of the MIS where the topology type is stored.

unique_name

```
const RWCString& unique_name() const;

void unique_name(
    const RWCString& unique_name
);
```

The name of the topology type. This identifier is unique within a single MIS.

8.16.5 General Member Functions

make_null

```
void make_null();
```

Sets to *null* value. A *null* value means that the EMTopoTypeDn does not refer to any topology type.

`is_null`

```
RWBoolean is_null() const;
```

And, test for *null* value. A *null* value means that the `EMTopoTypeDn` does not refer to any topology type.

8.17 EMTopoType Class

Inheritance: `EMObject`

```
#include topo_api/topo_api.hh
```

An instance of the `EMTopoType` class represents a topology type. Every topology node is classified as a particular topology type. The topology types form a hierarchy with the six base types “Container”, “Device”, “Monitor”, “Link”, “Array” and “Bus” with other subtypes derived from them. Beyond the standard POC methods which allow you to create, delete, compare, etc. topology types, the `EMTopoType` class provides the following additional services:

- static methods `is_container()`, `is_device()`, `is_monitor()`, `is_link()`, `is_view()`, `is_array`, and `is_bus` can be used to categorize topology types.

The EM topo type attributes are described in TABLE 8-5.

The attribute key is:

- C – Attribute can be set at creation time.
- S – Attribute can be set after creation time.
- M – Mandatory; attribute must be set for operation to succeed.
- X – Allowed; attribute can be set as an option.

TABLE 8-5 EM TopoType Attributes

Attribute Enum	C	S	Description
<code>dn</code>	M	M	Unique identifier.
<code>base_type</code>	M		The parent topology type of this type.
<code>all_base_types</code>			All ancestors of this type.
<code>sub_types</code>			All topology types contained by this type.

TABLE 8-5 EM TopoType Attributes (*Continued*)

Attribute Enum	C	S	Description
legal_children	X	X	List of legal topology types of topology nodes that can be contained by a topology node of this type within the topology hierarchy.
layer_name	M	X	Name of the layer that includes topology nodes of this type.
user_data_attribute_names	X	X	A list of GDMO attribute names that define the contents of the EMTopoNode::user_data attribute for EMTopoNodes of this type.

8.17.1 Example

```

RWBoolean
create_topo_type(
    const RWCString& system_name,
    const RWCString& type_name
)
{
    EMTopoType type(EMTopoTypeDn(system_name,type_name));

    type.set_base_type(EMTopoTypeDn::device);
    type.set_layer_name(type_name);

    EMStatus status;
    if (!(status = type.create_with_all_attributes())) {
        cerr << "Error: " << status << endl;
        return FALSE;
    }
    return TRUE;
}

```

Enum

```
enum EMTopoType::Attribute {  
    dn=0,  
    base_type,  
    sub_types,  
    legal_children,  
    layer_name,  
    user_data_attribute_names,  
    num_attributes  
};
```

8.17.2 Constructors and Destructor

```
EMTopoType(  
    const EMTopoTypeDn& dn  
);
```

```
EMTopoType(  
    const EMTopoType& topo_type  
);
```

The destructor is,

```
~EMTopoType();
```


8.17.3 Operators

```
EMTopoType& operator =(
    const EMTopoType& topo_type
);
```

8.17.4 Access Member Functions

The member function methods of the `EMTopoType` class refer to the attributes listed in TABLE 8-5. See the table for a description of each of the attributes.

`get_dn`

```
EMStatus get_dn(
    EMTopoTypeDn& dn
) const;
```

This function gets the distinguished name, *dn*, associated with a node.

`set_dn`

```
EMStatus set_dn(
    const EMTopoTypeDn& dn
);
```

This function sets the distinguished name, *dn*, associated with a node.

`get_base_type`

```
EMStatus get_base_type(
    RWCString& type_name
) const;
```

This function gets the parent topology type of this type.

set_base_type

```
EMStatus set_base_type(  
    const RWCString& type_name  
);
```

This function sets the parent topology type of this type.

get_all_base_types

```
EMStatus get_all_base_types(  
    RWTValSlistRWCString& base_types  
) const;
```

This function gets all the ancestors of this type.

get_sub_types

```
EMStatus get_sub_types(  
    RWTValSlistRWCString& sub_types  
) const;
```

This function gets all the topology types contained by this type.

get_legal_children

```
EMStatus get_legal_children(  
    RWTValSlistRWCString& children  
) const;
```

This function gets a list of legal topology types of topology nodes that can be contained by a topology node of this type within the topology hierarchy.

add_legal_child

```
EMStatus add_legal_child(  
    const RWCString& child  
);
```

This function adds *child* to the list of legal topology types of topology nodes that can be contained by a topology node of this type within the topology hierarchy.

get_layer_name

```
EMStatus get_layer_name(  
    RWCString& layer_name  
) const;
```

This function gets the name of the layer that includes topology nodes of this type.

set_layer_name

```
EMStatus set_layer_name(  
    const RWCString& layer_name  
);
```

This function sets the name of the layer that includes topology nodes of this type.

get_user_data_attribute_names

```
EMStatus get_user_data_attribute_names(  
    RWTValSlistRWCString& names  
) const;
```

This function gets a list of GDMO attribute names that define the contents of the `EMTopoNode::user_data` attribute for `EMTopoNodes` of this type.

set_user_data_attribute_names

```
EMStatus set_user_data_attribute_names(  
    const RWTValSlistRWCString& names  
);
```

This function sets a list of GDMO attribute names that define the contents of the `EMTopoNode::user_data` attribute for `EMTopoNodes` of this type.

add_user_data_attribute_name

```
EMStatus add_user_data_attribute_name(  
    const RWCString& name  
);
```

This function adds *name* to the list of GDMO attribute names that define the contents of the `EMTopoNode::user_data` attribute for `EMTopoNodes` of this type.

remove_user_data_attribute_name

```
EMStatus remove_user_data_attribute_name(  
    const RWCString& name  
);
```

This function removes *name* from the list of GDMO attribute names that define the contents of the `EMTopoNode::user_data` attribute for `EMTopoNodes` of this type.

8.17.5 Static Member Functions

These methods return `TRUE`, if *type_name* is a subtype of the indicated base type.

`is_container`

```
static RWBoolean is_container(  
    const RWCString& type_name  
);
```

This function returns `TRUE`, if *type_name* is a subtype of the container type.

`is_monitor`

```
static RWBoolean is_monitor(  
    const RWCString& type_name  
);
```

This function returns `TRUE`, if *type_name* is a subtype of the monitor type.

`is_view`

```
static RWBoolean is_view(  
    const RWCString& type_name  
);
```

This function returns `TRUE`, if *type_name* is a view. A type is a view if nodes of the type can contain other nodes, and all the view attributes are supported. For example, all containers and monitors are considered views.

The method `is_view()` is special because there is no base type named 'View'; `is_view()` is equivalent to `is_container()` or `is_monitor()`.

is_device

```
static RWBoolean is_device(  
    const RWCString& type_name  
);
```

This function returns TRUE, if *type_name* is a subtype of the device type.

is_link

```
static RWBoolean is_link(  
    const RWCString& type_name  
);
```

This function returns TRUE, if *type_name* is a subtype of the link type.

is_array

```
static RWBoolean is_array(  
    const RWCString& type_name  
);
```

This function returns TRUE, if *type_name* is a subtype of the link type.

is_bus

```
static RWBoolean is_bus(  
    const RWCString& type_name  
);
```

This function returns TRUE, if *type_name* is a subtype of the link type.

8.17.6 Static Member Functions for Event Subscription

The `EMTopoType` class provides an event subscription service so that clients can be notified when a topology type is created, deleted, or modified.

```
struct EMTopoTypeCallbackData
{
//
// Note: em_change_event is not supported at this time.
//
    EMEventType event_type;
    EMTopoTypeDn node_dn
    EMTopoNode changes;
    void* client_data;
};

typedef void (*Callback)(
    const EMTopoTypeCallbackData& cbd
);
```

To register for events, the client must provide a callback function with of type `EMTopoType::Callback`. When the client's callback is called, the *cbd* parameter will be filled in with information about the event. The *event_type* field indicates the type of event: `em_create_event`, or `em_delete_event`.

The *node_dn* parameter uniquely identifies the topology type that was created, deleted, or modified. For `em_change_event` only, the *changes* parameter will contain the new values for all attributes which changed. To get a list of the changed attributes, call `EMTopoType::get_active_attributes()`. The normal `EMTopoType` access methods may be used to get the new attribute values. Finally, the *client_data* field is the same as the *client_data* parameter of `EMTopoType::register_callback()`.

```
void register_callback(
    EMEventType event,
    Callback callback,
    void* client_data
);
```

Registers *callback* to be called when *event* occurs on any topology type. If *event* equals `em_any_event`, then *callback* will be called for any of `em_create_event`, `em_delete_event`, or `em_change_event`. The parameter *client_data* will be used to initialize the *client_data* field in the `EMTopoTypeCallbackData` struct.

Note – If the same *callback* has already been registered for the same *event*, then the callback will not be added a second time. However, the *client_data* will replace the previous *client_data*.

```
void unregister_callback(  
    EMEventType event,  
    EMTopoType::Callback callback  
);
```

Removes *callback* that was previously registered for *event* events.

Note – If *callback* was registered multiple times with different *event* parameters, the *callback* will only be removed for this *event*.

8.17.7 Global Operators

```
ostream& operator<<(  
    ostream& s,  
    const EMTopoType& type  
);
```

The stream output operator `<<` is defined to provide an easy way to print out the attribute values of `EMTopoType`.

8.18 EMAgent Class

Inheritance: EMOBJECT

```
#include topo_api/topo_api.hh
```

The EMAgent class is an abstract class that contains the agent interface, common between EMCmpAgent, EMRpcAgent, and EMSnmpAgent.

Enum

```
enum EMAgent::Attribute {
    operational_state,
    administrative_state,
    num_attributes
};
```

```
enum EMAgent::AdministrativeState {
    locked,
    unlocked,
    shuttingdown,
    num_administrative_states
};
```

Used to suspend and resume the proxy activity relative to the Internet Agent. The EMAgent::unlocked state means that the proxy must continue to perform, or resume performing, proxy activities on behalf of the Internet agent. The EMAgent::locked state means that the proxy must not perform, or suspend performing, proxy activities on behalf of the Internet agent.

```
enum EMAgent::OperationalState {
    disabled,
    enabled,
    num_operational_states
};
```

Indicates the perceived state of the Internet agent. The `EMAgent::enabled` state means that the Internet agent is operational, as perceived by the proxy: it can be reached. The `EMAgent::disabled` state means that the Internet agent is not operational, as perceived by the proxy; it cannot be reached.

8.18.1 Access Member Functions

`get_operational_state`

```
EMStatus get_operational_state(  
    OperationalState& operational_state  
) const;
```

This function gets the operational state of a component of an equipment from among the managed objects on a network; the possible values are `EMAgent::disabled` or `EMAgent::enabled`.

Note – The `operational_state` is read-only.

`get_administrative_state`

```
EMStatus get_administrative_state(  
    AdministrativeState& administrative_state  
) const;
```

This function gets the administrative state of a component of an equipment from among the managed objects on a network; the possible values are `EMAgent::locked`, `unlocked`, or `shuttingdown`.

set_administrative_state

```
EMStatus set_administrative_state(
    const AdministrativeState& administrative_state
);
```

This function sets the administrative state of a component of an equipment from among the managed objects on a network; the possible values are `EMAgent::locked`, `unlocked`, or `shuttingdown`.

8.19 EMCmipAgentDn Class

Inheritance: none

```
#include topo_api/topo_api.hh
```

An instance of the `EMCmipAgentDn` class uniquely identifies one rpc agent object out of the set of rpc agent objects interfaced by the `EMCmipAgent` persistent object class.

8.19.1 Constructors and Destructor

```
EMCmipAgentDn();
```

The default constructor creates a null object.

```
EMCmipAgentDn(
    const RWCString& system_name,
    const RWCString& unique_name
);
```

The above constructor takes the MIS name where the object is stored and the CMIP agent name.

And the default destructor,

```
~EMCmipAgentDn();
```

8.19.2 Operators

```
RWBoolean operator ==(
    const EMCmipAgentDn& dn
) const;
```

Two instances are equal if they have both the same system name and the same unique name, or if they are both Null.

And the *not*-operator,

```
RWBoolean operator !=(
    const EMTopoNodeDn& dn
) const;
```

8.19.3 Access Member Functions

system_name

```
const RWCString& system_name() const;

void system_name(
    const RWCString& system_name
);
```

The name of the MIS where the CMIP agent object is stored.

unique_name

```
const RWCString& unique_name() const;

void unique_name(
    const RWCString& unique_name
);
```

The name of the CMIP agent object. This name is unique within a single MIS.

8.19.4 General Member Functions

make_null

```
void make_null();
RWBoolean is_null() const;
```

Sets to null value and tests for null value. A null value means that the EMCmipAgentDn does not refer to any cmip agent object.

8.19.5 Related Global Operators

```
ostream& operator<<(ostream& s, const EMTopoNodeDn& dn);
```

The stream output operator << is defined to provide an easy way to print out the value of EMTopoNodeDn.

8.20 EMCmipAgent Class

Inheritance: EMAgent <- EMOBJECT

```
#include topo_api/topo_api.hh
```

An instance of the EMCmipAgent class represents the MIS object which contains configuration information for a CMIP agent. The configuration information includes the CMIP MPA hostname and port number, list of managed objects DNs, network SAP, transport selector, presentation selector, session selector, and application entity title (AET).

Note – This class does not provide an interface to the agent’s managed objects, but only to Solstice EM’s configuration information for the agent.

TABLE 8-6 gives the CmpipAgent attributes.

The attribute key is:

- C – Attribute can be set at creation time.
- S – Attribute can be set after creation time.
- M – Mandatory; attribute must be set for operation to succeed.
- X – Allowed; attribute can be set as an option.

TABLE 8-6 EMCmipAgent Attributes

Attribute Enum	C	S	Description
dn	M	M	Unique identifier
operational_state			EMAgent::disabled or enabled
administrative_state	M	X	EMAgent::locked, unlocked, or shuttingdown
mpa_address_info	X	X	MPA hostname and port number
agent_address_tag	X	X	Defines format of agent_address_info
agent_address_info	M	X	Agent address information in format defined by agent_address_tag.

TABLE 8-6 EMCmipAgent Attributes (*Continued*)

Attribute Enum	C	S	Description
managed_objects	M	X	List of DN's in ASCII slash format of managed objects located on agent. Note that the multiple cmip agent configurations can be created for the same cmip MPA but with a different set of managed objects for each.
application_entity_title	M	X	Application Entity Title (AET).
presentation_selector	M	X	OSI presentation selector.
session_selector	M	X	OSI session selector.
transport_selector	M	X	OSI transport selector.
network_sap	M	X	OSI network sap.
name_translation	X	X	Specifies the format of the managed object instance when sending a request to the CMIP agent. It can be LDN, FDN, or NONE. NONE means FDN. The default value is "NONE".
application_entity_qualifier	X	X	The Application Entity Qualifier. The default value is -1.
application_entity_invoke_id	X	X	The Application Entity Invocation Identifier. The default value is -1.
application_process_invoke_id	X	X	The Application Process Invocation Identifier. The default value is -1.

8.20.1 Example

```

setup_mis_mis_connection(
    const RWCString& manager_hostname,
    const RWCString& agent_hostname
)
{
    EMCmipAgent cmip_agent(EMCmipAgentDn(manager_hostname,
                                           agent_hostname));
    cmip_agent.set_administrative_state(EMAgent::unlocked);
    cmip_agent.set_mpa_address_info(agent_hostname, 5555);

    RWCString managed_object("/systemId=name:\"");
    managed_object.append(agent_hostname).append("\"");
    cmip_agent.add_managed_object(managed_object);
    cmip_agent.set_application_entity_title("objectIdentifier :
{ 1 2 3 4 }");
    cmip_agent.set_presentation_selector("");
    cmip_agent.set_session_selector("");
    cmip_agent.set_transport_selector("");
    char buffer[128];
    sprintf(buffer, "%s:%d", agent_hostname, 5555);
    cmip_agent.set_network_sap(buffer);
    // String: {psel,ssel,tssel,nsap}
    cmip_agent.set_agent_address_tag(8);
    sprintf(buffer, "{, , %s:%d", agent_hostname, 5555);
    cmip_agent.set_agent_address_info(buffer);

    EMStatus status;
    if (!(status = cmip_agent.create_with_all_attributes())) {
        cerr << "Error: " << status << endl;
    }
}

```


Enum

```
enum EMCmipAgent::Attribute {
    dn = EMAgent::num_attributes,
    mpa_address_info,
    agent_address_info,
    agent_address_tag,
    managed_objects,
    application_entity_title,
    presentation_selector,
    session_selector,
    transport_selector,
    network_sap,
    num_attributes
};
```

8.20.2 Access Member Functions

This function gets the distinguished name, *dn*, associated with a node.

The member function methods of the `EMCmipAgent` class refer to the attributes listed in TABLE 8-6. See the table for a description of each of the attributes.

get_dn

```
EMStatus get_dn(
    EMCmipAgentDn& dn
) const;
```

This function gets the distinguished name, *dn*, associated with an agent.

set_dn

```
EMStatus set_dn(
    const EMCmipAgentDn& dn
);
```

This function sets the distinguished name, *dn*, associated with an agent.

get_mpa_address_info

```
EMStatus get_mpa_address_info(  
    RWCString& hostname,  
    int& port_number,  
    RWBoolean& is_null  
    ) const;
```

This function gets the MPA hostname and port number.

set_mpa_address_info

```
EMStatus set_mpa_address_info(  
    const RWCString& hostname,  
    int port_number,  
    RWBoolean is_null = FALSE  
    );
```

This function sets the MPA hostname and port number.

get_managed_objects

```
EMStatus get_managed_objects(  
    RWTValSlistRWCString& dns  
    ) const;
```

This function gets a list of DNS in slash format of managed objects located on agent. Note that the multiple cmip agent configurations can be created for the same cmip MPA but with a different set of managed objects for each.

set_managed_objects

```
EMStatus set_managed_objects(  
    const RWTValSlistRWCString& dns  
);
```

This function sets a list of DNs in slash format of managed objects located on agent. Note that the multiple cmip agent configurations can be created for the same cmip MPA but with a different set of managed objects for each.

add_managed_object

```
EMStatus add_managed_object(  
    const RWCString& dn  
);
```

This function adds *dn* to the list of DNs in slash format of managed objects located on agent. Note that the multiple cmip agent configurations can be created for the same cmip MPA but with a different set of managed objects for each.

remove_managed_object

```
EMStatus remove_managed_object(  
    const RWCString& dn  
);
```

This function removes *dn* from the list of DNs in slash format of managed objects located on agent. Note that the multiple cmip agent configurations can be created for the same cmip MPA but with a different set of managed objects for each.

get_network_sap

```
EMStatus get_network_sap(  
    RWCString& network_sap  
) const;
```

This function gets the OSI network sap.

set_network_sap

```
EMStatus set_network_sap(  
    const RWCString& network_sap  
);
```

This function sets the OSI network sap.

get_agent_address_info

```
EMStatus get_agent_address_info(  
    RWCString& agent_address_info  
    ) const;
```

This function gets the agent address information in a format defined by `agent_address_tag`.

set_agent_address_info

```
EMStatus set_agent_address_info(  
    const RWCString& agent_address_info  
);
```

This function sets the agent address information in a format defined by `agent_address_tag`.

get_agent_address_tag

```
EMStatus get_agent_address_tag(  
    int& agent_address_tag  
    ) const;
```

This function gets the defined format of `agent_address_info`.

set_agent_address_tag

```
EMStatus set_agent_address_tag(  
    int agent_address_tag  
);
```

This function sets the defined format of agent_address_info.

get_presentation_selector

```
EMStatus get_presentation_selector(  
    RWCString& presentation_selector  
    ) const;
```

This function gets the OSI presentation selector.

set_presentation_selector

```
EMStatus set_presentation_selector(  
    const RWCString& presentation_selector  
    );
```

This function sets the OSI presentation selector.

get_session_selector

```
EMStatus get_session_selector(  
    RWCString& session_selector  
    ) const;
```

This function gets the OSI session selector.

set_session_selector

```
EMStatus set_session_selector(  
    const RWCString& session_selector  
);
```

This function sets the OSI session selector.

get_transport_selector

```
EMStatus get_transport_selector(  
    RWCString& transport_selector  
    ) const;
```

This function gets the OSI transport selector.

set_transport_selector

```
EMStatus set_transport_selector(  
    const RWCString& transport_selector  
);
```

This function sets the OSI transport selector.

get_application_entity_title

```
EMStatus get_application_entity_title(  
    RWCString& application_entity_title  
    ) const;
```

This function gets the application entity title (AET).

set_application_entity_title

```
EMStatus set_application_entity_title(  
    const RWCString& application_entity_title  
);
```

This function sets the application entity title (AET).

get_name_translation

```
EMStatus get_name_translation(  
    RWCString& name_translation  
) const;
```

This function gets the format of the managed object instance in a request to the CMIP agent.

set_name_translation

```
EMStatus set_name_translation(  
    const RWCString& name_translation  
);
```

This function sets the format of the managed object instance in a request to the CMIP agent.

get_application_entity_qualifier

```
EMStatus get_application_entity_qualifier(  
    RWCString& application_entity_qualifier  
) const;
```

This function gets the Application Entity Qualifier.

set_application_entity_qualifier

```
EMStatus set_application_entity_qualifier(  
    const RWCString& application_entity_qualifier  
);
```

This function sets the Application Entity Qualifier.

get_application_entity_invoke_id

```
EMStatus get_application_entity_invoke_id(  
    int& application_entity_invoke_id  
) const;
```

This function gets the Application Entity Invocation Identifier.

set_application_entity_invoke_id

```
EMStatus set_application_entity_invoke_id(  
    int application_entity_invoke_id  
);
```

This function sets the Application Entity Invocation Identifier.

get_application_process_invoke_id

```
EMStatus get_application_process_invoke_id(  
    int& application_process_invoke_id  
) const;
```

This function gets the Application Process Invocation Identifier.


```
set_application_process_invoke_id
```

```
EMStatus set_application_process_invoke_id(
    int application_process_invoke_id
);
```

This function sets the Application Process Invocation Identifier.

8.20.3 Global Operators

```
ostream& operator<<(ostream& s, const EMCmipAgent& agent);
```

The stream output operator << is defined to provide an easy way to print out the attribute values of EMCmipAgent.

8.21 EMRpcAgentDn Class

Inheritance: none

```
#include topo_api/topo_api.hh
```

An instance of the EMRpcAgentDn class uniquely identifies one rpc agent object out of the set of rpc agent objects interfaced by the EMRpcAgent persistent object class.

8.21.1 Constructors and Destructor

The following are constructors:

```
EMRpcAgentDn();
```

The default constructor creates a null object.

```
EMRpcAgentDn(  
    const RWCString& system_name,  
    const RWCString& unique_name  
);
```

The above constructor takes the MIS name where the object is stored and the rpc agent name.

And the default destructor,

```
~EMRpcAgentDn();
```

8.21.2 Operators

```
RWBoolean operator ==(  
    const EMRpcAgentDn& dn  
    ) const;
```

Two instances are equal if they have both the same system name and the same unique name or if they are both Null.

And the *not*-operator,

```
RWBoolean operator !=(  
    const EMRpcAgentDn& dn  
    ) const;
```

8.21.3 Access Member Functions

system_name

```
const RWCString& system_name() const;

void system_name(
    const RWCString& system_name
);
```

The name of the MIS where the rpc agent object is stored.

unique_name

```
const RWCString& unique_name() const;

void unique_name(
    const RWCString& unique_name
);
```

The name of the rpc agent object. This name is unique within a single MIS.

8.21.4 General Member Functions

make_null

```
void make_null();
```

Sets to null value. A null value means that the `EMRpcAgentDn` does not refer to any rpc agent object.

```
is_null
```

```
RWBoolean is_null() const;
```

And tests for null value.

8.21.5 Global Operators

```
ostream& operator<<(ostream& s, const EMRpcAgentDn& dn);
```

The stream output operator << is defined to provide an easy way to print out the value of EMRpcAgentDn.

8.22 EMRpcAgent Class

Inheritance: EAgent <- EMOject

```
#include topo_api/topo_api.hh
```

An instance of the EMRpcAgent class represents the MIS object which contains configuration information for an RPC agent. The configuration information includes the read and write community strings, and supported schemas.

This class does not provide an interface to the agent's managed objects, but only to Solstice EM's configuration information for the agent.

TABLE 8-7 gives the EMRpcAgent attributes. The attribute key is:

- C – Attribute can be set at creation time.
- S – Attribute can be set after creation time.
- M – Mandatory; attribute must be set for operation to succeed.
- X – Allowed; attribute can be set as an option

TABLE 8-7 EMRpcAgent Attributes

Attribute Enum	C	S	Description
dn	M	M	Unique identifier
operational_state			EMAgent::disabled or enabled
administrative_state	M	X	EMAgent::locked, unlocked, or shuttingdown
get_community_string	X	X	e.g. "public", "private"
set_community_string	X	X	e.g. "public", "private"
schemas	M	X	list of rpc_proxy_hostname and rpc_name pairs, e.g. "ultra-server", "RPC Proxy -ping"

8.22.1 Example

```

RWBoolean
create_rpc_agent(
    const RWCString& system_name,
    const RWCString& rpc_agent_name
)
{
    EMRpcAgent rpc_agent(EMRpcAgentDn(system_name,
                                     rpc_agent_name));
    rpc_agent.set_administrative_state(EMAgent::unlocked);
    EMRpcAgent::Schema schema("proxy-hostname",
                              "RPC Proxy -ping");
    rpc_agent.add_schema(schema);

    EMStatus status;
    if (!(status = rpc_agent.create_with_all_attributes())) {
        cerr << "Error: " << status << endl;
        return FALSE;
    }
    return TRUE;
}

```

Enum

```
enum EMRpcAgent::Attribute {
    dn = EMAgent::num_attributes,
    get_community_string,
    set_community_string,
    schemas,
    num_attributes
};
```

These are the attributes specific to EMRpcAgent, in addition to the attributes defined in EMAgent which are common to EMCmipAgent, EMRpcAgent, and EMSnmpAgent.

Struct

```
struct EMRpcAgent::Schema
{
    RWCString name;
    RWCString proxy_hostname;

    Schema();

    Schema(const RWCString& p_name,
           const RWCString& p_proxy_hostname);

    RWBoolean operator ==(const Schema& schema) const;

    RWBoolean operator !=(const Schema& schema) const;
}

ostream& operator<<(ostream& s, const Schema& schema);
```

The struct EMRpcAgent::Schema is used to store a RPC proxy hostname and RPC method pairing. Each EMRpcAgent can be configured to support any number of schemas.

8.22.2 Constructors and Destructor

```
EMRpcAgent (
    const EMRpcAgentDn& rpc_agent_dn
);
```

```
EMRpcAgent (
    const EMRpcAgent& rpc_agent
);
```

The default destructor,

```
~EMRpcAgent ();
```

8.22.3 Access Member Functions

The member function methods of the `EMRpcAgent` class refer to the attributes listed in the table at the beginning of this section, table 6-7. See the table for a description of each attribute.

`get_dn`

```
EMStatus get_dn(
    EMRpcAgentDn& dn
) const;
```

This function gets the distinguished name, *dn*, associated with an agent.

set_dn

```
EMStatus set_dn(  
    const EMRpcAgentDn& dn  
);
```

This function sets the distinguished name, *dn*, associated with an agent.

get_get_community_string

```
EMStatus get_get_community_string(  
    RWCString& get_community_string  
) const;
```

This function gets the current *get_community_string* codeword to a component of an equipment from the managed objects on a network, e.g. “public” or “private”.

set_get_community_string

```
EMStatus set_get_community_string(  
    const RWCString& get_community_string  
);
```

This function sets the current *get_community_string* codeword to a component of an equipment from the managed objects on a network, e.g. “public” or “private”.

get_set_community_string

```
EMStatus get_set_community_string(  
    RWCString& set_community_string  
) const;
```

This function gets the current *set_community_string* codeword to a component of an equipment from the managed objects on a network, e.g. “public” or “private”.

set_set_community_string

```
EMStatus set_set_community_string(
    const RWCString& set_community_string
);
```

This function sets the current *set_community_string* codeword to a component of an equipment from the managed objects on a network, e.g. “public” or “private”.

get_schemas

```
EMStatus get_schemas(
    RWTValSlistEMRpcAgent::Schema& schemas
) const;
```

This function gets the list of *rpc_proxy_hostname* and *rpc_name* pairs, e.g. “ultra-server”, “RPC Proxy -ping”.

set_schemas

```
EMStatus set_schemas(
    const RWTValSlistEMRpcAgent::Schema& schemas
);
```

This function sets the list of *rpc_proxy_hostname* and *rpc_name* pairs, e.g. “ultra-server”, “RPC Proxy -ping”.

add_schema

```
EMStatus add_schema(
    const EMRpcAgent::Schema& schema
);
```

This function adds *schema* to the list of *rpc_proxy_hostname* and *rpc_name* pairs, e.g. “ultra-server”, “RPC Proxy -ping”.

remove_schema

```
EMStatus remove_schema(
    const EMRpcAgent::Schema& schema
);
```

This function removes *schema* from the list of `rpc_proxy_hostname` and `rpc_name` pairs, e.g. “ultra-server”, “RPC Proxy -ping”.

8.22.4 Global Operators

```
ostream& operator<<(
    ostream& s,
    const EMRpcAgent& agent
);
```

The stream output operator `<<` is defined to provide an easy way to print out the attribute values of `EMRpcAgent`.

The Assignment Operator

```
EMRpcAgent& operator =(
    const EMRpcAgent&
);
```

8.23 EMSnmpAgentDn Class

Inheritance: none

```
#include topo_api/topo_api.hh
```

An instance of the `EMSnmpAgentDn` class uniquely identifies one `snmp` agent object out of the set of `snmp` agent objects interfaced by the `EMSnmpAgent` persistent object class.

8.23.1 Constructors, and Destructor

The following are constructors for EMSnmpAgentDn

```
EMSnmpAgentDn();
```

The default constructor creates a null object.

```
EMSnmpAgentDn(  
    const RWCString& system_name,  
    const RWCString& unique_name  
);
```

The above constructor takes the MIS name where the object is stored and the SNMP agent name.

8.23.2 Operators

```
RWBoolean operator ==(  
    const EMSnmpAgentDn& dn  
) const;  
  
RWBoolean operator !=(  
    const EMSnmpAgentDn& dn  
) const;
```

Two instances are equal if they have both the same system name and the same unique name or if they are both null.

8.23.3 Access Member Functions

`system_name`

```
const RWCString& system_name() const;

void system_name(
    const RWCString& system_name
);
```

The name of the MIS where the snmp agent object is stored.

`unique_name`

```
const RWCString& unique_name() const;

void unique_name(
    const RWCString& unique_name
);
```

The name of the SNMP agent object which is unique on one MIS. Combined with the `system_name`, the pair form a globally unique identifier.

8.23.4 General Member Functions

`make_null`

```
void make_null();
```

Sets to null value. A null value means that the `EMSnmpAgentDn` does not refer to any SNMP agent object.

```
is_null
```

```
RWBoolean is_null() const;
```

And tests for the null value.

8.23.5 Global Operators

```
ostream& operator<<(
    ostream& s,
    const EMSnmpAgentDn& dn
);
```

The stream output operator<< is defined to provide an easy way to print out the value of EMSnmpAgentDn.

8.24 EMSnmpAgent Class

Inheritance: EAgent <- EObject

```
#include topo_api/topo_api.hh
```

An instance of the EMSnmpAgent class represents the MIS object that contains configuration information for an SNMP agent. The configuration information includes the read and write community strings, supported MIBs, and transport address.

Note – This class does not provide an interface to the agent’s managed objects, only to Solstice EM’s configuration information for the agent.

TABLE 8-8 gives the EMSnmpAgent attributes.

The attribute key is:

- C – Attribute can be set at creation time.
- S – Attribute can be set after creation time.
- M – Mandatory; attribute must be set for operation to succeed.
- X – Allowed; attribute can be set as an option.

TABLE 8-8 EMSnmpAgent Attributes

Attribute Enum	C	S	Description
dn	M	M	Unique identifier, which includes the administrative name.
operational_state	X		Possible values are EMAgent::disabled and EMAgent::enabled.
administrative_state	M	X	Possible values are EMAgent::locked, EMAgent::unlocked, or EMAgent::shuttingdown.
system_title	M		OID of system title, e.g. "1.2.3.4".
get_community_string	M	X	e.g. "public" or "private".
set_community_string	M	X	e.g. "public" or "private".
transport_address	M	X	IP address of the system associated with the Internet agent, specified as a string, such as "34.254.129.23". An optional port number may be appended, such as "34.254.129.23:5723".
management_protocol	M		Internet management protocol used by the proxy to manage devices. Possible values are: EMSnmpAgent::snmp_v1 and EMSnmpAgent::snmp_v2.
supported_mibs	M	X	The names of the MIBs that the SNMP agent supports.
access_control_enforcement	M		Indicates where access control is applied: at the Internet agent, the ISO/Internet proxy, or both. Possible values are EMSnmpAgent::agent, EMSnmpAgent::proxy, or EMSnmpAgent::both.
access_control_mechanism	X		Indicates whether no access control, Internet access control as specified in [SNMPv2SEC], or ISO/CCITT access control as specified in [ISO10164-9] is to be used. Possible values are EMSnmpAgent::no_access_control, EMSnmpAgent::internet, or EMSnmpAgent::iso.

8.24.1 Example

```

RWBoolean
create_snmp_agent(
    const RWCString& system_name,
    const RWCString& snmp_agent_name
)
{
    EMSnmAgent snmp_agent(
        EMSnmAgentDn(system_name, snmp_agent_name));
    snmp_agent.set_administrative_state(EMAgent::unlocked);
    snmp_agent.add_supported_mib("IIMCRFC1213-MIB");
    snmp_agent.add_supported_mib("IIMCSUN-MIB");
    snmp_agent.set_get_community_string("public");
    snmp_agent.set_set_community_string("private");
    snmp_agent.set_transport_address("123.234.34.23:2354");
    snmp_agent.set_management_protocol(EMSnmAgent::snmp_v1);
    snmp_agent.set_access_control_enforcement(
        EMSnmAgent::agent);
    snmp_agent.set_access_control_mechanism(
        EMSnmAgent::internet);
    snmp_agent.set_system_title("1.2.3.4");
    EMStatus status;
    if (!(status = snmp_agent.create_with_all_attributes())) {
        cerr << "Error: " << status << endl;
        return FALSE;
    }
    return TRUE;
}

```

Enum

```

enum EMSnmAgent::Attribute {
    dn = EMAgent::num_attributes,
    system_title,
    get_community_string,
    set_community_string,
    transport_address,
    supported_mibs,
    management_protocol,
    access_control_enforcement,
    access_control_mechanism,
    num_attributes
};

```

These are the attributes specific to EMSnmpAgent, in addition to the attribute defined in EMAgent which are common to EMCmipAgent, EMRpcAgent, and EMSnmpAgent.

```
enum EMSnmpAgent::AccessControlEnforcement {
    agent=1,
    proxy=2,
    both=3,
    min_access_control_enforcement = agent,
    max_access_control_enforcement = both,
    num_access_control_enforcements
};

ostream& operator<< (
    ostream& s,
    const EMSnmpAgent::AccessControlEnforcement& enforcement
);
```

```
enum EMSnmpAgent::AccessControlMechanism {
    no_access_control=0,
    internet=1,
    iso=2,
    min_access_control_mechanism = no_access_control,
    max_access_control_mechanism = iso,
    num_access_control_mechanisms
};

ostream& operator<< (
    ostream& s,
    const EMSnmpAgent::AccessControlMechanism& mechanism
);
```



```

enum EMSnpAgent::ManagementProtocol {
    snmp_v1,
    snmp_v2,
    num_management_protocols
};

ostream& operator<<(
    ostream& s,
    const EMSnpAgent::ManagementProtocol& protocol
);

```

8.24.2 Constructors and Destructor

```

EMSnpAgent(
    const EMSnpAgentDn& snmp_agent_id
);

```

```

EMSnpAgent(
    const EMSnpAgent& snmp_agent
);

```

And the destructor,

```

~EMSnpAgent();
EMSnpAgent& operator =(
    const EMSnpAgent& other_agent
);

```

8.24.3 Access Member Functions

The member function methods of the `EMSnmpAgent` class all refer to the attributes listed in the table at the beginning of this section, TABLE 8-8. See the table for a description of each attribute.

`get_dn`

```
EMStatus get_dn(  
    EMSnmpAgentDn& dn  
) const;
```

This function gets a unique identifier, the distinguished name *dn*, which includes the administrative name.

`set_dn`

```
EMStatus set_dn(  
    const EMSnmpAgentDn& dn  
);
```

This function sets a unique identifier, the distinguished name *dn*, which includes the administrative name.

`get_system_title`

```
EMStatus get_system_title(  
    RWCString& system_title  
) const;
```

This function gets the OID of the system, or system title, for example, “1.2.3.4”.

`set_system_title`

```
EMStatus set_system_title(
    const RWCString& system_title
);
```

This function sets the OID of the system, or system title, for example, “1.2.3.4”.

`get_get_community_string`

```
EMStatus get_get_community_string(
    RWCString& get_community_string
) const;
```

This function gets the current *get_community_string* to a component of an equipment from the managed objects on a network, e.g. “public” or “private”.

`set_get_community_string`

```
EMStatus set_get_community_string(
    const RWCString& get_community_string
);
```

This function sets the current *get_community_string* to a component of an equipment from the managed objects on a network, e.g. “public” or “private”.

`get_set_community_string`

```
EMStatus get_set_community_string(
    RWCString& set_community_string
) const;
```

This function gets the current *set_community_string* to a component of an equipment from the managed objects on a network, e.g. “public” or “private”.

set_set_community_string

```
EMStatus set_set_community_string(  
    const RWCString& set_community_string  
);
```

This function sets the current *set_community_string* to a component of an equipment from the managed objects on a network, e.g. “public” or “private”.

get_transport_address

```
EMStatus get_transport_address(  
    RWCString& transport_address  
) const;
```

This function gets the IP address of the system associated with the Internet agent, specified as a string, such as “34.254.129.23”. An optional port number may be appended, such as “34.254.129.23:5723”.

set_transport_address

```
EMStatus set_transport_address(  
    const RWCString& transport_address  
);
```

This function sets the IP address of the system associated with the Internet agent, specified as a string, such as “34.254.129.23”. An optional port number may be appended, such as “34.254.129.23:5723”.

get_supported_mibs

```
EMStatus get_supported_mibs(  
    RWTValSlistRWCString& supported_mibs  
) const;
```

This function gets the names of the MIBs that the SNMP agent supports.

set_supported_mibs

```
EMStatus set_supported_mibs(  
    const RWTValSlistRWCString& supported_mibs  
);
```

This function sets the names of the MIBs that the SNMP agent supports.

add_supported_mib

```
EMStatus add_supported_mib(  
    const RWCString& supported_mib  
);
```

This function adds *supported_mib* to the names of the MIBs that the SNMP agent supports.

remove_supported_mib

```
EMStatus remove_supported_mib(  
    const RWCString& supported_mib  
);
```

This function removes *supported_mib* from the names of the MIBs that the SNMP agent supports.

get_management_protocol

```
EMStatus get_management_protocol(  
    EMSnmpAgent::ManagementProtocol& management_protocol  
) const;
```

This function gets the Internet management protocol used by the proxy to manage devices. Possible values are EMSnmpAgent::snmp_v1 and EMSnmpAgent::snmp_v2.

set_management_protocol

```
EMStatus set_management_protocol(  
    EMSnmpAgent::ManagementProtocol management_protocol  
);
```

This function sets the Internet management protocol used by the proxy to manage devices. Possible values are `EMSnmpAgent::snmp_v1` and `EMSnmpAgent::snmp_v2`.

get_access_control_enforcement

```
EMStatus get_access_control_enforcement(  
    EMSnmpAgent::AccessControlEnforcement&  
        access_control_enforcement  
    ) const;
```

This function gets and indicates where access control is applied: at the Internet agent, the ISO/Internet proxy, or both. Possible values are `EMSnmpAgent::agent`, `EMSnmpAgent::proxy`, or `EMSnmpAgent::both`.

set_access_control_enforcement

```
EMStatus set_access_control_enforcement(  
    EMSnmpAgent::AccessControlEnforcement  
        access_control_enforcement  
    );
```

This function sets and indicates where access control is applied: at the Internet agent, the ISO/Internet proxy, or both. Possible values are `EMSnmpAgent::agent`, `EMSnmpAgent::proxy`, or `EMSnmpAgent::both`.

get_access_control_mechanism

```
EMStatus get_access_control_mechanism(
    EMSnmpAgent::AccessControlMechanism&
        access_control_mechanism
    ) const;
```

This function gets and indicates whether no access control, Internet access control as specified in [SNMPv2SEC], or ISO/CCITT access control as specified in [ISO10164-9] is to be used. Possible values are `EMSnmpAgent::no_access_control`, `EMSnmpAgent::internet`, or `EMSnmpAgent::iso`.

set_access_control_mechanism

```
EMStatus set_access_control_mechanism(
    EMSnmpAgent::AccessControlMechanism
        access_control_mechanism
    );
```

This function sets and Indicates whether no access control, Internet access control as specified in [SNMPv2SEC], or ISO/CCITT access control as specified in [ISO10164-9] is to be used. Possible values are `EMSnmpAgent::no_access_control`, `EMSnmpAgent::internet`, or `EMSnmpAgent::iso`.

8.24.4 Related Global Operators

```
ostream& operator<<(
    ostream& s,
    const EMSnmpAgent& agent
    );
```

The stream output operator `<<` is defined to provide an easy way to print out the value of `EMSnmpAgent`.

Object Services API

The Object Services API, or object development tools (ODT), allow developers who implement object classes to send CMIS requests in an ODT application. This API is useful for developing manager and agent network management applications. Developers can use it to customize behaviors for the GDMO classes, rather than accept the default behaviors.

The ODT do not allow users to extend or override any existing services or object behavior within the MIS, such as the NerveCenter or Event Service.

This chapter comprises the following topics:

- Section 9.1 “Operational Flow” on page 9-2
- Section 9.2 “Service Request Function Parameters” on page 9-3
- Section 9.3 “Service Response Callback Function Parameters” on page 9-7
- Section 9.4 “Services Interface Descriptions and Examples” on page 9-7
- Section 9.5 “Supporting Functions for Example Code” on page 9-57

9.1 Operational Flow

The Object Services API provides a set of programming interfaces for use by an application developer when writing object behavior software. An application developer is not required to use the services functions. However, these functions make it easier to perform some common tasks related to inter-object communication from within.

The operational flow of a request issued using the Object Services API is based on the PMI `Message` and `MessageSAP` C++ classes on object behavior. The `Message` and `MessageSAP` C++ classes are used throughout the MIS and are also the basis for the low level PMI interface. The Object Services API hides the `Message` classes and the `MessageSAP` classes used by the low level PMI. The classes are hidden primarily to simplify this API and also for the following reasons:

- Most messages sent through this API can use a number of default values. The Object Services API function calls provide default values for all parameters not specifically required for a particular operation.
- Only a single well defined `MessageSAP` is required for these functions. The services function calls involve communication between the object access module (OAM) and the message routing module (MRM). The OAM contains both user-developed object behavior code and the generated object behavior code. The MRM handles routing for all message requests and responses.

9.2 Service Request Function Parameters

TABLE 9-1 provides detailed descriptions of the Service Request Function parameters defined for the Object Services API.

TABLE 9-1 Service Request Function Parameters

Parameter	Description
<code>const Asn1Value oc</code>	<p>Instance of the <code>Asn1Value</code> C++ class that contains an Object Identifier (OID) for a managed object class. For each of the Object Service API request functions, except the <code>send_event_req</code> function, this parameter can be the OID for Actual Class. Actual Class is an ISO defined OID that matches the class of any managed object on which an operation is performed. The <i>oc</i> parameter is used as follows:</p> <ul style="list-style-type: none">• <i>Single Object Selection (Base Object Only Scoping):</i> The OID specifies the object class of the managed object from which attribute values are retrieved, using the Get operation.• <i>Multiple Object Selection Using Scoping and Filtering:</i> The <i>OID parameter</i> specifies the object class of the managed object used as the starting point for the selection of managed objects from which attribute values are retrieved using the Get operation. <p>The CMIS and CMIP specifications refer to this parameter as either the base object class or managed object class, depending on the type of operation being performed.</p>
<code>const Asn1Value oi</code>	<p>Instance of the <code>Asn1Value</code> C++ class that contains either a distinguished name (context specific 2) or a local distinguished name (context specific 4) for a managed object instance. The <i>oi parameter</i> is used as follows:</p> <ul style="list-style-type: none">• <i>Single Object Selection (Base Object Only Scoping):</i> The <i>oi parameter</i> specifies the name of the managed object instance from which the request operation is performed.• <i>Multiple Object Selection Using Scoping and Filtering:</i> The <i>oi parameter</i> specifies the name of the managed object instance to be used as the starting point for the selection of managed objects on which the request operation is performed. <p>The CMIS and CMIP specifications refer to this parameter as either the base object class or managed object instance, depending on the type of operation being performed. Specify this parameter as a null <code>Asn1Value</code> (<code>Asn1Value()</code>) if a managed object instance name is specified by the <i>superior_oi parameter</i>.</p>

TABLE 9-1 Service Request Function Parameters (*Continued*)

Parameter	Description
<code>const Callback cb</code>	Instance of the <code>Callback</code> C++ class, which can contain two pointers. The first is a pointer of a callback function to be invoked when a response to an operation is received. The second is a pointer to application developer-specified data (commonly referred to as user data) to be passed to the callback function when it is invoked. The user data pointer is always optional. The <code>cb</code> parameter is required, or optional for service request functions that support it. If specified, a <i>confirmed</i> service request is issued. If not specified, an <i>unconfirmed</i> service request is issued.
<code>const Asn1Value attr_list</code>	Instance of the <code>Asn1Value</code> C++ class that contains a set of OIDs for attributes. Attributes are normally members of the object class specified by the <i>oc</i> parameter or members of the class of an object instance identified within a scoped operation.
<code>const Asn1Value modify_list</code>	A set of attribute ID and attribute value pairs that specify for a <code>send_set_req</code> service request operation which attributes are to be modified and new values for those attributes. The <i>modify_list</i> parameter is an instance of the <code>Asn1Value</code> C++ class and is typically a sequence of an OID that identifies an attribute, followed by a value for the attribute.
<code>const Asn1Value action_type</code>	Instance of the <code>Asn1Value</code> C++ class that contains an object identifier (OID) that specifies the type of action generated by the <code>send_action_req</code> function.
<code>const Asn1Value action_info</code>	Instance of the <code>Asn1Value</code> C++ class that contains any event information associated with the type of action specified by the <i>action_type</i> parameter. The <i>action_info</i> parameter typically contains a sequence or set of ASN.1-defined values. The <i>action_info</i> parameter is optional for the <code>send_action_req</code> service function but must be present if a <code>WITH INFORMATION SYNTAX</code> construct is specified as part of the GDMO definition for the action type specified by the <i>action_type</i> parameter.

TABLE 9-1 Service Request Function Parameters (*Continued*)

Parameter	Description
<code>const Asn1Value attr_value_list</code>	A set of attribute ID and attribute value pairs that specify the attributes assigned new values by a <code>send_create_req</code> service request operation. The values specified in the <code>send_create_req</code> service function override the corresponding attributes from the reference object (if specified using the <i>reference_oi parameter</i>) or from the default value specified in the GDMO definition for the managed object class. The <i>attr_value_list parameter</i> is an instance of the <code>Asn1Value</code> C++ class and is typically a sequence of an OID that identifies an attribute, followed by a value for the attribute. The <i>attr_value_list parameter</i> is an optional parameter for the <code>send_create_req</code> service request, although values must be specified for all mandatory attributes defined in the GDMO managed object class definition for which an instance is being created. In other words, the mandatory attribute values must be specified in the GDMO definition in a default value clause, supplied from a reference object, or else specified in the <code>attr_value_list</code> .
<code>const Asn1Value superior_oi</code>	An instance of the <code>Asn1Value</code> C++ class, this parameter is used only with the <code>send_create_req</code> function. The parameter contains either a distinguished name (context specific 2) or a local distinguished name (context specific 4) for an existing managed object instance that is to be the superior—in the MIT—of the managed object instance created. This parameter should not be specified, or should be specified as a null <code>Asn1Value</code> , if a managed object instance name is specified by the <i>oi parameter</i> .
<code>const Asn1Value reference_oi</code>	An instance of the <code>Asn1Value</code> C++ class, this parameter is used only with the <code>send_create_req</code> function. It contains either a distinguished name (context specific 2) or a local distinguished name (context specific 4) for an existing managed object instance that is of the same class as the managed object instance created. Attribute values associated with the managed object specified by the <i>reference_oi parameter</i> become default values for those attributes not specified by the <i>attr_value_list parameter</i> of the <code>send_create_req</code> function.
<code>const Asn1Value event_type</code>	Instance of the <code>Asn1Value</code> C++ class that contains an OID specifying the type of notification to be generated by the <code>send_event_req</code> function.

TABLE 9-1 Service Request Function Parameters (*Continued*)

Parameter	Description
<code>const Asn1Value event_info</code>	An <code>Asn1Value</code> C++ class that contains any event information associated with the type of notification specified by the <i>event_type</i> parameter. This parameter typically contains a sequence or set of ASN.1-defined values and is optional for the <code>send_event_req</code> service function. It must be present if a <code>WITH INFORMATION SYNTAX</code> construct is specified as part of the GDMO definition for the notification type specified by the <i>event_type</i> parameter.
<code>const Asn1Value event_time</code>	Instance of the <code>Asn1Value</code> C++ class containing a value for the time at which a notification is generated.
<code>MessId id</code>	Identifier that uniquely identifies a service request operation. If specified, the value for this parameter is generated and set by the service request function.
<code>const MessScope scope</code>	Specifies the type of scoping used for a service request operation. The possible types of scoping are <code>BASE_OBJECT</code> , <code>NTH_LEVEL</code> , <code>BASE-TO-NTH_Level</code> , and <code>ALL_LEVELS</code> . It is optional for all service request functions that support it. If not specified, it defaults to <code>BASE_OBJECT</code> .
<code>const Asn1Value filter</code>	Instance of the <code>Asn1Value</code> C++ class that contains a <code>CMISFilter</code> (refer to the ISO DMI for a definition of the CMIS filter). All objects selected by the scoping parameter are tested against a filter. The service request operation is performed only on those objects that pass the filter test. It is an optional parameter for all service request functions that support it. If not specified, it defaults to a filter that matches all managed objects selected by the <i>scope</i> parameter.
<code>const MessSync sync</code>	Specifies the type of synchronization used for a service request operation. It is an enumerated type and can take on the value <code>ATOMIC</code> or <code>BEST_EFFORT</code> . It is an optional parameter for all service request functions that support it. If not specified, it defaults to <code>BEST_EFFORT</code> . The use of <code>ATOMIC</code> is rarely supported by remote objects.
<code>const Asn1Value access</code>	A reserved parameter not normally specified. The <i>access</i> parameter defines the access control that objects selected for a service request operation must pass. The service request operation is not performed by any managed object that does not pass the access control.

9.3 Service Response Callback Function Parameters

TABLE 9-2 describes the Service Response Callback Function Parameters.

TABLE 9-2 Service Response Callback Function Parameters

Parameter	Description
<code>Ptr userdata</code>	Optional parameter intended for all service response callback functions. It is a <code>void *</code> pointer to data specified by the application developer in the callback parameter of a service request function.
<code>Ptr message</code>	Mandatory parameter for all service response callback functions. It is a <code>void *</code> pointer to the message generated in response to a service request function.

9.4 Services Interface Descriptions and Examples

The following service request functions and service indication functions are supported by the Object Services API. In general, the services interfaces specified here contain both mandatory and optional parameters. Mandatory parameters are typically passed by reference. Optional parameters are typically passed either by value or by a pointer to a value. Mandatory parameters are ordered prior to the optional parameters for each function. Optional parameters for each function are ordered by placing the most-likely-to-be-specified optional parameters first.

This section includes the following functions:

- Get Request Service
- Get Response Callback
- Set Request Service
- Set Response Callback
- Action Request Service
- Action Response Callback
- Create Request Service

- Create Response Callback
- Delete Request Service
- Delete Response Callback
- Delete Response Callback Parameter Description
- Event Report Request Service (Unconfirmed)
- Event Report Response Callback

9.4.1 Get Request Service

Enables you to override GDMO restrictions that are related to managed object (MO) attribute getting.

The following subsections describe and provide examples of the Get Request Service.

9.4.1.1 Get Request Service

The rules for getting an MO's attributes are defined in the MO class definition in a GDMO document. The rules impose restrictions on getting attributes, and it is possible to override some of these restrictions when requesting the getting of an MO's attributes through the `send_get_req()` function. Specifically, the absence of a GET or GET-REPLACE modifier for an attribute is intended to have the effect of restricting the getting of an attribute by a management application.

To allow behavior internal to the MIS to override this restriction, this function is defined with a `flags` parameter. The relevant flag value is `ReqMess::OVERRIDE_ATTR_CHECKS`, which is defined in `message.hh`.

Note – In the case where a scope and filter are specified, the override capability only applies to the base object, and not to the scoped and filtered objects.

9.4.1.2 Interface Signature

```
Result send_get_req(
    const Asn1Value &oc,
    const Asn1Value &oi,
    const Callback &cb,    // Always confirmed
    const Asn1Value attr_list = Asn1Value(),
                                // Default: Get all attributes

    MessId *id = 0,
    const MessScope scope = MessScope(),
                                //Default: Base object only
    const Asn1Value filter = Asn1Value(), //Default: Matches all
    const MessSync sync = BEST_EFFORT,
    const Asn1Value access = Asn1Value(),
    Boolean sub_trans
); //Default: No access cntrl
```

9.4.1.3 send_get_request Parameter Descriptions

TABLE 9-3 describes the `send_get_request` parameters.

TABLE 9-3 send_get_request Parameters

Parameter	Description	Required/ Optional
const Asn1Value <i>oc</i>	Specifies the class of the base managed object.	Required
const Asn1Value <i>oi</i>	Distinguished name of local distinguished name for the base managed object.	Required
const Callback <i>cb</i>	Specifies the name of a callback function invoked when a response to the get request is received and can be used to specify user data that is passed to the callback function when it is invoked.	Required
const Asn1Value <i>attr_list</i>	List of attribute OIDs whose attribute values are to be returned in response to the get request operation. If not specified, all attributes defined for the managed object class are returned in the response.	Optional
MessId <i>id</i>	Provides a unique identifier for a particular Get request operation. If specified, the value for this parameter is generated and set by the <code>send_get_req</code> function.	Optional

TABLE 9-3 `send_get_request` Parameters (*Continued*)

Parameter	Description	Required/ Optional
<code>const MessScope scope</code>	Defines the type of scoping used for this request operation. If not specified, it defaults to <code>BASE_OBJECT_ONLY</code> .	Optional
<code>const Asn1Value filter</code>	Defines a filter to be passed by all objects selected by the scoping parameter. If not specified, it defaults to a filter that matches all managed objects selected by the <code>scope</code> parameter.	Optional
<code>const MessSync sync</code>	Defines the type of synchronization used for this request operation. If not specified, it defaults to <code>BEST_EFFORT</code> .	Optional
<code>Boolean sub_trans</code>	Spawn subtransactions instead of creating new transactions. If <code>sub_trans</code> is not specified, the default behavior is to create a new transaction for the operation.	Optional
<code>const Asn1Value access</code>	Reserved parameter that should not be used at this time.	Not available

9.4.1.4 send_get_request Examples

Get Request: Base Object Only

CODE EXAMPLE 9-1 Get Request: Base Object Only Example

```

Result
get_system_object_attributes(char *system_name)
{
    Asn1Value actual_oc;
    Asn1Value system_oi;

    //*****
    // (Confirmed) Get request
    //*****
    // Send a get request to an instance of the
    // system managed object class. The system
    // the request is sent to is identified by
    // the system_name parameter.

    VTRY {

        // Encode oc OID. In this example, actualClass is used
        // instead of the OID for the system managed object
        // class.
        // Note: CMIP requires a TAG_CONT(0) encoding for the
        // object class rather than TAG_OID (refer to x711.asn1
        // for encoding spec).

        TTRYRES(actual_oc.encode_oid(TAG_CONT(0),
            Oid("2.9.3.4.3.42")));

        // Encode the distinguished name for an instance of the
        // system managed object class. Either the distinguished
        // name form (TAG_CONT(2)) or the local distinguished name
        // form (TAG_CONT(4)) can be used with the get operation.
        // The distinguished name form is used in this example.
        // The get_sys_dn function is included in Section 9.5.

        TTRYRES(get_sys_dn(system_name, system_oi));

        // In this example, all attributes from the system object
class
        // are retrieved. No attribute ID list is required.

        // Send the get request (Always Confirmed). No user data is

```

CODE EXAMPLE 9-1 Get Request: Base Object Only Example (*Continued*)

```
// specified for the callback parameter. The get_req_cb
function
// is included as part of the Object Services API examples.

    objsvc_test.print("About to issue Get Request\n");
    if (send_get_req(actual_oc, system_oi,
        Callback(get_req_cb, 0)) != OK) {
        objsvc_test.print("Error issuing Get Request\n");
        return NOT_OK;
    }
    else
        objsvc_test.print("Issued Get Request\n");
    }
VBEGHANDLERS
VCATCHALL {
    objsvc_test.print("\nError encoding Get Request\n");
    return NOT_OK;
}
VENDHANDLERS
return OK;
}
```

Get Request: Scoped Operation

CODE EXAMPLE 9-2 Get Request: Scoped Operation Example

```
Result
get_application_attributes(char *system_name)
{
    Asn1Value actual_oc;
    Asn1Value system_oi;
    Asn1Value em_mis_rdn;

    //*****
    // (Confirmed) Scoped get request
    //*****
    // Get the emApplicationID, emApplicationType, and the
    // emUserID attribute values for each application instance
    // object under an emKernel object. The emKernel object
    // is located under the system object in the MIT.

    VTRY {

        // Encode oc OID. In this example, actualClass is used
```

CODE EXAMPLE 9-2 Get Request: Scoped Operation Example (*Continued*)

```

// instead of the OID for the system managed object
// class.
// Note: CMIP requires a TAG_CONT(0) encoding for the
// object class rather than TAG_OID (refer to x711.asn1
// for encoding spec).

TTRYRES(actual_oc.encode_oid(TAG_CONT(0),
    Oid("2.9.3.4.3.42")));

// Encode the distinguished name for an instance of the
// emKernel managed object class. Either the distinguished
// name form (TAG_CONT(2)) or the local distinguished name
// form (TAG_CONT(4)) can be used with the get operation.
// The distinguished name form is used in this example.
// The get_sys_dn and get_graphstr_rdn functions are included
// in Section 9.5.

TTRYRES(get_sys_dn(system_name, system_oi));
TTRYRES(get_graphstr_rdn(
    "2.9.3.5.7.11", "EM-MIS", em_mis_rdn));
TTRYRES(system_oi.add_component(em_mis_rdn));

// Encode the attribute list. The CMIP spec specifies
// TAG_CONT(12) as the tag for the attribute list.

Asn1Value attrlist;
Asn1Value enc_oid1, enc_oid2, enc_oid6;
Oid oid1("1.3.6.1.4.1.42.2.2.1.7.1"); // emApplicationID
Oid oid2("1.3.6.1.4.1.42.2.2.1.7.2"); // emApplicationType
Oid oid6("1.3.6.1.4.1.42.2.2.1.7.6"); // emUserID
TTRYRES(attrlist.start_construct(TAG_CONT(12)));
TTRYRES(enc_oid1.encode_oid(TAG_CONT(0), oid1));
TTRYRES(enc_oid2.encode_oid(TAG_CONT(0), oid2));
TTRYRES(enc_oid6.encode_oid(TAG_CONT(0), oid6));
TTRYRES(attrlist.add_component(enc_oid1));
TTRYRES(attrlist.add_component(enc_oid2));
TTRYRES(attrlist.add_component(enc_oid6));

// Send the scoped get request (Always Confirmed). No user
data
// is specified for the callback parameter. The
// scoped_get_req_cb function is included as part of the
// Object Services API examples.

objsvc_test.print("About to issue Scoped Get Request\n");
if (send_get_req(actual_oc, system_oi,

```

CODE EXAMPLE 9-2 Get Request: Scoped Operation Example (*Continued*)

```
        Callback(scoped_get_req_cb, 0), attrlist,
        0, MessScope(NTH_LEVEL, 1)) != OK) {
    objsvc_test.print("Error issuing Scoped Get
Request\n");
    return NOT_OK;
}
else
    objsvc_test.print("Issued Scoped Get Request\n");
}
VBEGHANDLERS
VCATCHALL {
    objsvc_test.print("\nError encoding Get Request\n");
    return NOT_OK;
}
VENDHANDLERS
return OK;
}
```

9.4.2 Get Response Callback

The `send_get_req` service function requires a callback function. The name of the callback function provided by the application developer must match the name of the callback function specified in the `Callback` parameter of the `send_get_req` function.

9.4.2.1 Interface Signature

```
void user-provided-get-response-callback(
    Ptr userdata,          // Pointer to user supplied data
    Ptr message);          // Pointer to GetRes message
```

9.4.2.2 Get Response Callback Parameter Descriptions

TABLE 9-4 shows the Get response callback parameters.

TABLE 9-4 Get Response Callback Parameters

Parameter	Description
<i>userdata</i>	A void * pointer to data specified by the application developer in the callback parameter of the <code>send_get_req</code> function.
<i>message</i>	A void * pointer to the GetRes message generated in response to the <code>send_get_req</code> function.

9.4.2.3 Get Response Callback Examples

Callback Function: Single Response (Base Object Only)

CODE EXAMPLE 9-3 Callback Function: Single Response

```
void
get_req_cb(Ptr, Ptr get_response_msg)
{
    objsvc_test.print("Get Request callback\n");
    Message *resp = (Message *)get_response_msg;

    VTRY {
        if ( resp->type() == GET_RES) {
            GetRes *g_resp = (GetRes *)resp;
            g_resp->print(objsvc_test);

            // The AsnValue decoding functions, including the
            // member functions get_first_component and
            // get_next_component can be used to examine the member data
            // included in the response message at this point

            // ...

        }
        else if ( resp->is_error() ) {
            objsvc_test.print("Get response error received\n");
            objsvc_test.print( "message type = %s\n",
                               MessType_fmt(resp->type()));
            resp->print(objsvc_error);
        }
        else {
            objsvc_test.print(
                "Unexpected or invalid response received\n");
            objsvc_test.print( "message type = %s\n",
                               MessType_fmt(resp->type()));
            resp->print(objsvc_error);
        }
    }
    VBEGHANDLERS
    VCATCHALL {
        objsvc_test.print("Error processing response for Get\n");
        resp->print(objsvc_error);
    }
    VENDHANDLERS
}
```


CODE EXAMPLE 9-3 Callback Function: Single Response (*Continued*)

```

    if ( resp )
        Message:: delete_message(resp);
}

```

Callback Function: Multiple Responses (Scoped Operation)**CODE EXAMPLE 9-4** Callback Function: Multiple Responses

```

static int resp_count = 0;
static int err_count = 0;
static int unknown_count = 0;
void
scoped_get_req_cb(Ptr userdata, Ptr calldata)
{
    objsvc_test.print("Scoped Get Request callback:");
    Message *resp = (Message *)calldata;

    VTRY {
        if ( resp->type() == GET_RES) {
            GetRes *sg_resp = (GetRes *)resp;
            if (sg_resp->linked) {
                objsvc_test.print("**** LINKED Response ****\n");
                resp_count++;
                sg_resp->print(objsvc_test);

                // The Asn1Value decoding functions, including the
                // member functions get_first_component and
                // get_next_component can be used to examine the member
                // data included in the response message here.

                // ...

            } else {
                objsvc_test.print("**** Final Response ****\n");
                sg_resp->print(objsvc_test);
                objsvc_test.print("Valid: %d, Error: %d, Invalid: %d\n",
                    resp_count, err_count, unknown_count);

                // Final response processing can be performed here. The
                // final response message does not contain any attribute
                // value data

                // ...
            }
        }
    }
}

```

CODE EXAMPLE 9-4 Callback Function: Multiple Responses (*Continued*)

```

    }
}
else if ( resp->is_error() ) {
    objsvc_test.print("Error Response\n");
    objsvc_test.print( "message type = %s\n",
        MessageType_fmt(resp->type()));
    resp->print(objsvc_error);
    err_count++;
}
else {
    objsvc_test.print("Invalid Message\n");
    objsvc_test.print( "message type = %s\n",
        MessageType_fmt(resp->type()));
    resp->print(objsvc_error);
    unknown_count++;
}
}
}
VBEGHANDLERS
VCATCHALL {
    objsvc_test.print("\nError processing response for Scoped
Get\n");
}
VENDHANDLERS

if ( resp )
    Message:: delete_message(resp);
}

```

9.4.3 Set Request Service

Enables you to override GDMO restrictions that are related to MO attribute setting.

This section describes the interface signature, `send_set_request` parameters, and provides an example of its use.

9.4.3.1 Set Request Service

The rules for setting an MO's attributes are defined in the MO class definition in a GDMO document. The rules impose restrictions on the setting of attributes, but it is possible to override some of these restrictions when requesting the setting of an MO's attributes through the `send_set_req()` function. Specifically, the absence of a `REPLACE`, `GET-REPLACE`, or `ADD-REMOVE` modifier for an attribute is intended to have the effect of restricting the setting of an attribute by a management application.

To allow behavior internal to the MIS to override this restriction, this function is defined with a `flags` parameter. The relevant flag value is

`ReqMess::OVERRIDE_ATTR_CHECKS`, which is defined in `message.hh`.

Note – In the case where a scope and filter are specified the override capability only applies to the base object, and not to the scoped and filtered objects.

9.4.3.2 Interface Signature

CODE EXAMPLE 9-5 Set Request Interface Signature

```
Result send_set_req(  
    const Asn1Value &oc,  
    const Asn1Value &oi,  
    const Asn1Value &modify_list,  
    const Callback cb = 0, // Default: Unconfirmed  
    MessId *id = 0,  
    const MessScope scope = MessScope(),  
                                // Default: Is base object only  
    const Asn1Value filter = Asn1Value(), // Default: Matches all  
    const MessSync sync = BEST_EFFORT,  
    const Asn1Value access = Asn1Value(),  
    Boolean sub_trans); // Default: No access cntrl
```

9.4.3.3 send_set_request Parameter Descriptions

TABLE 9-5 provides descriptions of the send_set_request parameters.

TABLE 9-5 send_set_request Parameters

Parameter	Description	Required/ Optional
const Asn1Value oc	OID that specifies the class of the base managed object.	Required
const Asn1Value oi	Distinguished name or local distinguished name for the base managed object.	Required
const Asn1Value modify_list	List of attribute ID and attribute value pairs that specify which attributes are to be modified by the set request operation and what the new values are for the attributes.	Required
const Callback cb	Specifies the name of a callback function invoked when a response to the set request operation is received. Can also be used to specify user data passed to the callback function when it is invoked. If specified, a confirmed set request operation is issued and the callback function is invoked when the response is received. If not specified, an unconfirmed set request operation is issued and no response is generated.	Optional
MessId id	Provides a unique identifier for a particular set request operation. If specified, the value for this parameter is generated and set by the send_set_req function.	Optional

TABLE 9-5 send_set_request Parameters (*Continued*)

Parameter	Description	Required/ Optional
const MessScope <i>scope</i>	Defines the type of scoping used for this request operation. If not specified, it defaults to BASE_OBJECT_ONLY.	Optional
const AsnlValue <i>filter</i>	Defines a filter to be passed by all objects selected by the scoping parameter. If not specified, it defaults to a filter that matches all managed objects selected by the <i>scope</i> parameter.	Optional
const MessSync <i>sync</i>	Defines the type of synchronization used for this request operation. If not specified, it defaults to BEST_EFFORT.	Optional
Boolean <i>sub_trans</i>	Spawn subtransactions instead of creating new transactions. If <i>sub_trans</i> is not specified, the default behavior is to create a new transaction for the operation.	Optional
const AsnlValue <i>access</i>	This parameter is reserved at this time.	Not available

9.4.3.4 send_set_request Example

CODE EXAMPLE 9-6 send_set_request

```

Result
set_log_adminState(char *system_name)
{
    Asn1Value actual_oc;
    Asn1Value log_fdn;
    Asn1Value log_rdn;
    Asn1Value mod_list;

    //*****
    // Confirmed Set request
    //*****
    // Send a confirmed set request to an instance of the
    // log managed object class. The log that
    // the request is sent to is
    // contained under the system identified by
    // the system_name parameter.

    VTRY {

        // Encode oc OID. In this example, actualClass is used
        // instead of the OID for the log managed object
        // class.
        // Note: CMIP requires a TAG_CONT(0) encoding for the
        // object class rather than TAG_OID (refer to x711.asn1
        // for encoding spec).

        TTRYRES(actual_oc.encode_oid(TAG_CONT(0),
            Oid("2.9.3.4.3.42")));

        // Encode the distinguished name for an instance of the
        // log managed object class. The instance logId="AlarmLog"
        // is used here. Either the distinguished name form
        // (TAG_CONT(2)) or the local distinguished name form
        // (TAG_CONT(4)) can be used with the set operation. The
        // distinguished name form is used in this example.
        // The get_sys_dn and get_graphstr_rdn functions are included
        // in Section 9.5.

        TTRYRES(get_sys_dn(system_name, log_fdn));
        TTRYRES(get_graphstr_rdn(Oid("2.9.3.2.7.2"), "AlarmLog",
            log_rdn));
        TTRYRES(log_fdn.add_component(log_rdn));
    }
}

```

CODE EXAMPLE 9-6 send_set_request (Continued)

```

// Encode the modification list for the set operation.
// The administrativeState of the log object is set to
// locked in this example.
// The CMIP spec specifies TAG_CONT(12) as the tag for
// the attribute modification list.

TTRYRES(modlist.start_construct(TAG_CONT(12)));
Asn1Value av;
TTRYRES(av.encode_enum(TAG_ENUM, 0)); // Set to LOCKED
Asn1Value set;//1 is unlocked
Asn1Value comp;//set the sequence of attr. ID and value pair
Oid oid("2.9.3.2.7.31"); // Set administrativeState
TTRYRES(set.start_construct(TAG_SEQ));
TTRYRES(comp.encode_oid(TAG_CONT(0), oid));
TTRYRES(set.add_component(comp));
TTRYRES(set.add_component(av));
TTRYRES(modlist.add_component(set));

// Send the confirmed set request. No user data is
// specified for the callback parameter. The set_req_cb
function
// is included as part of the Object Services API examples.

oamsvc_test.print("About to issue Confirmed Set Request\n");
if (send_set_req(actual_oc, log_fdn, modlist,
    Callback(set_req_cb, userdata)) != OK)
    objsvc_test.print("Error issuing Confirmed Set
Request\n");
    return NOT_OK;
}
else
    objsvc_test.print("Issued Confirmed Set Request\n");
}
VBEGHANDLERS
VCATCHALL {
objsvc_test.print("\nError encoding Set Request\n");
return NOT_OK;
}
VENDHANDLERS
return OK;
}

```

9.4.4 Set Response Callback

This callback function is used only in conjunction with confirmed Set requests. The name of the callback function must match the name of the callback function specified in the callback parameter of the `send_set_req` function.

9.4.4.1 Interface Signature

```
void user-provided-set-response-callback(  
    Ptr userdata,           // Pointer to user supplied data  
    Ptr message);          // Pointer to SetRes message
```

9.4.4.2 Set Response Callback Parameter Description

TABLE 9-6 describes the Set Response Callback parameters.

TABLE 9-6 Set Response Callback Parameters

Parameter	Description
Ptr <i>userdata</i>	A void * pointer to data specified by the application developer in the callback parameter of the <code>send_set_req</code> function.
Ptr <i>message</i>	A void * pointer to the SetRes message generated in response to the <code>send_set_req</code> function.

9.4.4.3 Set Response Callback Example

CODE EXAMPLE 9-7 Set Response Example

```

void
set_req_cb(Ptr, Ptr set_response_msg)
{
    objsvc_test.print("Set Request callback\n");
    Message *resp = (Message *)set_response_msg;

    VTRY {
        if ( resp->type() == SET_RES) {
            SetRes *s_resp = (SetRes *)resp;
            s_resp->print(objsvc_test);

            // The Asn1Value decoding functions, including the
            // member functions get_first_component and
            // get_next_component can be used to examine the member data
            // included in the response message at this point

            // ...

        }
        else if ( resp->is_error() ) {
            objsvc_test.print("Set response error received\n");
            objsvc_test.print( "message type = %s\n",
                               MessType_fmt(resp->type()));
            resp->print(objsvc_error);
        }
        else {
            objsvc_test.print(
                "Unexpected or invalid response received\n");
            objsvc_test.print( "message type = %s\n",
                               MessType_fmt(resp->type()));
            resp->print(objsvc_error);
        }
    }
    VBEGHANDLERS
    VCATCHALL {
        oamsvc_test.print("Error processing response for Set\n");
    }
    VENDHANDLERS

    if ( resp )
        Message:: delete_message(resp);
}

```

9.4.5 Action Request Service

9.4.5.1 Interface Signature

CODE EXAMPLE 9-8 Interface Signature for Action Request Service

```
Result send_action_req(
    const Asn1Value &oc,
    const Asn1Value &oi,
    const Asn1Value &action_type,
    const Asn1Value action_info = Asn1Value(), // Default: No action Info
    const Callback cb = 0, // Default: Unconfirmed
    MessId *id = 0,
    const MessScope scope = MessScope(), // Default: Base object Only
    const Asn1Value filter = Asn1Value(), // Default: Matches all
    const MessSync sync = BEST_EFFORT,
    const Asn1Value access = Asn1Value()); // Default: No access Cntrl
```

9.4.5.2 send_action_req Parameter Descriptions

TABLE 9-7 describes the send_action_req parameters.

TABLE 9-7 send_action_req Parameters

Parameter	Description	Required/ Optional
const Asn1Value <i>oc</i>	OID that specifies the class of the base managed object.	Required
const Asn1Value <i>oi</i>	Distinguished name or local distinguished name for the base managed object.	Required
const Asn1Value <i>action_type</i>	OID that specifies the type of action to be performed by the send_action_req function.	Required
const Asn1Value <i>action_info</i>	Contains information associated with the type of action specified by the <i>action_type</i> parameter. This option must be specified if a WITH INFORMATION SYNTAX construct is part of the GDMO definition for the action type specified by the <i>action_type</i> parameter.	Optional

TABLE 9-7 `send_action_req` Parameters (*Continued*)

Parameter	Description	Required/ Optional
<code>const Callback cb</code>	Specifies the name of a callback function invoked when a response to the action request operation is received and can be used to specify user data that is passed to the callback function when it is invoked. The callback parameter is optional for the <code>send_action_req</code> function. If specified, a confirmed action request operation is issued and the callback function is invoked when the response is received. If not specified, an unconfirmed action request operation is issued and no response is generated.	Optional
<code>MessId id</code>	Provides a unique identifier for a particular action request operation. If specified, the value for this parameter is generated and set by the <code>send_action_req</code> function.	Optional
<code>const MessScope scope</code>	Defines the type of scoping used for this request operation. If not specified, it defaults to <code>BASE_OBJECT_ONLY</code> .	Optional
<code>const AsnlValue filter</code>	Defines a filter to be passed by all objects selected by the scoping parameter. If not specified, it defaults to a filter that matches all managed objects selected by the scope parameter.	Optional
<code>const MessSync sync</code>	Defines the type of synchronization used for this request operation. If not specified, it defaults to <code>BEST_EFFORT</code> .	Optional
<code>const AsnlValue access</code>	Reserved parameter that should not be used at this time.	Not available

9.4.5.3 send_action_req Example

CODE EXAMPLE 9-9 send_action_req

```

Result
send_registerLocal_action(char *local_sys, char *remote_sys)
{
    Asn1Value actual_oc;
    Asn1Value dalarm_fdn;
    Asn1Value em_mis_rdn;
    Asn1Value dalarm_rdn;
    Asn1Value act_type;
    Asn1Value act_info;
    char      local_sys_id[100];
    char      remote_sys_id[100];
    Asn1Value local_id;
    Asn1Value remote_id;
    Asn1Value log_id;

    //*****
    // Confirmed Action request
    //*****
    // Send a confirmed registerLocal action request to an
    // instance of the distributed alarm manager object class.
    // The distributed alarm log manager that the request
    // is sent to is contained under the system identified
    // by the remote_sys parameter. The local_sys parameter
    // is the name of the system that contains the distributed
    // alarm log manager issuing this request.

    VTRY {

        // Encode oc OID. In this example, actualClass is used
        // instead of the OID for the log managed object
        // class.
        // Note: CMIP requires a TAG_CONT(0) encoding for the
        // object class rather than TAG_OID (refer to x711.asn1
        // for encoding spec).

        TTRYRES(actual_oc.encode_oid(TAG_CONT(0),
            Oid("2.9.3.4.3.42")));

        // Encode the distinguished name for an instance of the
        // distributed alarm log manager managed object class. The
        // distributed alarm log manager is always named
        // "Distrib_AlarmLog"
        // Either the distinguished name form (TAG_CONT(2))

```

CODE EXAMPLE 9-9 send_action_req (Continued)

```

// or the local distinguished name form (TAG_CONT(4)) can be
// used with the action operation. The distinguished name form
// is used in this example.
// The get_sys_dn and get_graphstr_rdn functions are included
// in Section 9.5.

TTRYRES(get_sys_dn(remote_sys, dalarm_fdn));
TTRYRES(get_graphstr_rdn("2.9.3.5.7.11",
    "EM-MIS", em_mis_rdn));
TTRYRES(get_graphstr_rdn(
    Oid("1.3.6.1.4.1.42.2.2.2.300.7.1", "Distrib-AlarmLog",
    dalarm_rdn));
TTRYRES(dalarm_fdn.add_component(em_mis_rdn));
TTRYRES(dalarm_fdn.add_component(dalarm_rdn));

// Encode the action type and action info parameters
// for the action operation. The ASN.1 for the registerLocal
// action_info is:
// LocalRegistrar ::= SEQUENCE
// {
//     receiverMIS      SystemId,
//     senderMIS        SystemId,
//     logId             SimpleNameType
// }
// The SystemIds for each of the systems also include
// a port number -- 5555 in this example. Port 5555
// is the default CMIP/LPP port used by the MIS.
// The CMIP spec specifies TAG_CONT(2) as the Tag for
// the action type.
TTRYRES(act_type.encode_oid(TAG_CONT(2),
    Oid("1.3.6.1.4.1.42.2.2.2.300.9.4")));

sprintf(local_sys_id, "%s:%s", local_sys, "5555");
TTRYRES(local_id.encode_octets(TAG_GRAPHSTR,
    DataUnit((char *)&local_sys_id)));
sprintf(remote_sys_id, "%s:%s", remote_sys, "5555");
TTRYRES(remote_id.encode_octets(TAG_GRAPHSTR,
    DataUnit((char *)&remote_sys_id)));
TTRYRES(log_id.encode_octets(TAG_GRAPHSTR, "AlarmLog"));

TTRYRES(act_info.start_construct(TAG_SEQ));
TTRYRES(act_info.add_component(remote_id));
TTRYRES(act_info.add_component(local_id));
TTRYRES(act_info.add_component(log_id));

// Send the confirmed action request. No user data is

```

CODE EXAMPLE 9-9 `send_action_req` (Continued)

```
// specified for the callback parameter. The action_req_cb
// function is included as part of the services
// API examples.

oamsvc_test.print("About to issue Confirmed Action
    Request\n");
if (send_action_req(actual_oc, dalarm_fdn, act_type,
    act_info, Callback(action_req_cb, 0)) != OK)
    objsvc_test.print("Error issuing Confirmed Action
        Request\n");
    return NOT_OK;
}
else
    objsvc_test.print("Issued Confirmed Action Request\n");
}
VBEGHANDLERS
VCATCHALL {
objsvc_test.print("\nError encoding Action Request\n");
return NOT_OK;
}
VENDHANDLERS
return OK;
}
```

9.4.6 Action Response Callback

This callback function is used only in conjunction with confirmed action requests. The name of the callback function must match the name of the callback function specified in the callback parameter of the `send_action_req` function.

9.4.6.1 Interface Signature

```
void user-provided-action-response-callback(
    Ptr userdata,          // Pointer to user supplied data
    Ptr message);          // Pointer to ActionRes message
```

9.4.6.2 Action Response Callback Parameter Description

TABLE 9-8 describes the action response callback parameters.

TABLE 9-8 Action Response Callback Parameters

Parameter	Description
Ptr <i>userdata</i>	A void * pointer to data specified by the application developer in the callback parameter of the <code>send_action_req</code> function.
Ptr <i>message</i>	A void * pointer to the ActionRes message generated in response to the <code>send_action_req</code> function.

9.4.6.3 Action Response Callback Example

The header file `dalarm.hh` is needed to successfully compile and link this example. The header file is needed to obtain the type definition for the `RegState` data type.

CODE EXAMPLE 9-10 Action Response Callback

```
void
action_req_cb(Ptr, Ptr action_response_msg)
{
    objsvc_test.print("Action Request callback\n");
    Message *resp = (Message *)action_response_msg;

    VTRY {
        if ( resp->type() == Action_RES) {
            ActionRes *a_resp = (ActionRes *)resp;
            a_resp->print(objsvc_test);

            // Decode the action reply field
            // LocalRegistrarReply ::= SEQUENCE
            // {
```

CODE EXAMPLE 9-10 Action Response Callback (*Continued*)

```

//      senderMIS      SystemId,
//      logId          SimpleNameType,
//      regStatus      RegistrationState
// }

    AsnlValue   remote_id, log_id, reg_state;
    DataUnit    sndr_du, log_du;
    I32         reg_val;
    RegState    reg;

    TTRYRES(a_resp->action_reply.
        first_component(remote_id));
    TTRYRES(remote_id.decode_octets(sndr_du));
    TTRYRES(a_resp->action_reply.next_component(
        remote_id, log_id));
    TTRYRES(log.decode_octets(log_du));
    TTRYRES(a_resp->action_reply.next_component(
        log_id, reg_state));
    TTRYRES(reg_state.decode_enum(reg_val));
    reg = (RegState)reg_val;

// Now do any other action response processing.
// ...

}
else if ( resp->is_error() ) {
    objsvc_test.print("Action response error received\n");
    objsvc_test.print( "message type = %s\n",
        MessType_fmt(resp->type()));
    resp->print(objsvc_error);
}
else {
    objsvc_test.print(
        "Unexpected or invalid response received\n");
    objsvc_test.print( "message type = %s\n",
        MessType_fmt(resp->type()));
    resp->print(objsvc_error);
}
}
}
VBEGHANDLERS
VCATCHALL {
oamsvc_test.print("Error processing response for Action\n");
}
VENDHANDLERS

```


CODE EXAMPLE 9-10 Action Response Callback (*Continued*)

```
if ( resp )
    Message:: delete_message( resp );
}
```

9.4.7 Create Request Service

Enables you to override GDMO restrictions that are related to MO creation.

9.4.7.1 Create Request Service

The rules for creating an MO are defined in the MO class definition and the name bindings in a GDMO document. The rules which are defined in the name bindings impose restrictions on the creation of MOs, but it is possible to override some of these restrictions when requesting creation of an MO through the function `send_create_req()`. Specifically, the absence of a `CREATE` construct in the name binding is intended to have the effect of restricting the creation of an MO by a management application.

To allow behavior internal to the MIS to override this restriction, this function is defined with a `flags` parameter. The relevant flag value is `ReqMess::OVERRIDE_NAME_BINDING`, which is defined in `message.hh`.

9.4.7.2 Interface Signature

```
Result send_create_req(
    const Asn1Value &oc,
    const Asn1Value &oi,
    const Asn1Value attr_value_list = Asn1Value(),
    const Callback cb = 0, // Default: Unconfirmed
    MessId *id = 0,
    const Asn1Value superior_oi = Asn1Value(),
    const Asn1Value reference_oi = Asn1Value(),
    const Asn1Value access = Asn1Value(),
    Boolean sub_trans
); / Default: No access ctrl
```

9.4.7.3 send_create_req Parameter Descriptions

TABLE 9-9 describes the send_create_req parameters.

TABLE 9-9 send_create_req Parameters

Parameter	Description	Required/ Optional
const Asn1Value <i>oc</i>	OID that specifies the class of the managed object.	Required
const Asn1Value <i>oi</i>	Distinguished name or local distinguished name for the managed object.	Required
const Asn1Value <i>attr_value_list</i>	Set of attribute ID and attribute value pairs that specify the attributes assigned new values by a send_create_req service request operation. The values specified in the send_create_req service function override the corresponding attributes from the reference object (if specified) or from the default value specified in the GDMO definition for the managed object class. Although this is an optional parameter, attribute values must be specified for all mandatory attributes defined for a GDMO managed object class, specified by a default value, by a reference object, or by this parameter.	Optional

TABLE 9-9 `send_create_req` Parameters (*Continued*)

Parameter	Description	Required/ Optional
<code>const Callback cb</code>	Specifies the name of a callback function invoked when a response to the create request operation is received. Can be used to specify user data passed to the callback function when it is invoked. Callback parameter is optional for the <code>send_create_req</code> function. If specified, a confirmed create request operation is issued and the callback function is invoked when the response is received. If not specified, an unconfirmed create request operation is issued and no response is generated.	Optional
<code>MessId id</code>	Provides a unique identifier for a particular create request operation. If specified, the value for this parameter is generated and set by the <code>send_create_req</code> function.	Optional
<code>const AsnlValue superior_oi</code>	Distinguished name or local distinguished name for an existing superior object under which a managed object is to be created. This optional parameter should not be specified if a value is specified for the <code>oi</code> parameter. The managed object created is contained under the superior object in the MIT.	Optional
<code>const AsnlValue reference_oi</code>	Distinguished name or local distinguished name for a reference object. If this parameter is specified, it must specify the name of a managed object of the same class as the managed object to be created. The attribute values of the reference object become default values for the new managed object for any attributes not specified in the value for the <code>attr_value_list</code> parameter.	Optional
<code>Boolean sub_trans</code>	Spawn subtransactions instead of creating new transactions. If <code>sub_trans</code> is not specified, the default behavior is to create a new transaction for the operation.	Optional
<code>const AsnlValue access</code>	Reserved parameter, not to be used at this time.	Not available

9.4.7.4 `send_create_req` Example

CODE EXAMPLE 9-11 shows how to override restrictions on creating an MO.

The `mysystem.gdmo` and `mysystem.asn1` files must be loaded into the MDR for this example to work properly. The `mysystem` files are located at `/opt/SUNWconn/em/src/scenario/example1`. The `README` file in this directory also describes how to load the `mysystem` files.

CODE EXAMPLE 9-11 `send_create_req`

```
Result
create_mySystem_object(char *system_name, char *mySystem_name)
{
    Asn1Value mySystem_oc;
    Asn1Value mySystem_fdn;
    Asn1Value mySystem_rdn;

    //*****
    // Confirmed create request
    //*****
    // Send a confirmed create request for an
    // instance of the mySystem manager object class.
    // Instances of the mySystem class are contained by
    // instances of the System class. The system_name
    // parameter specifies the name of the system to contain
    // the new instance. The mySystem_name parameter is used
    // to specify the name of the instance to be created.

    VTRY {

        // Encode oc OID. In this example, the OID for the
        // mySystem class is used.
        // Note: CMIP requires a TAG_CONT(0) encoding for the
        // object class rather than TAG_OID (refer to x711.asn1
        // for encoding spec).

        TTRYRES(mySystem_oc.encode_oid(
            TAG_CONT(0),Oid("1.2.3.4.5.6.3.10")));

        // Encode the distinguished name for an instance of the
        // mySystem managed object class. Instances of the mySystem
        // class are named using the systemId attribute in the class.
        // Either the distinguished name form (TAG_CONT(2))
        // or the local distinguished name form (TAG_CONT(4)) can be
        // used with the create operation. The distinguished name form
        // is used in this example.
        // The get_sys_dn and get_graphstr_rdn functions are included
        // in Section 9.5. The definition for the sys_id (systemId)
        // is included in Section 9.5.
    }
}
```

CODE EXAMPLE 9-11 send_create_req (Continued)

```

TTRYRES(get_sys_dn(system_name, mySystem_fdn));
TTRYRES(get_graphstr_rdn(sys_id, // systemId OID
    mySystem_name, mySystem_rdn));
TTRYRES(mySystem_fdn.add_component(mySystem_rdn));

// Encode the attribute list for the create request
// All mandatory attributes must be specified as part
// of the create request (unless the GDMO defines a
// initial value sub-clause for the attribute, or
// specifies a reference object that contains the
// attribute).
// Note: CMIP requires a TAG_CONT(7) encoding for the
// attribute list rather than TAG_SEQ (refer to x711.asn1
// for encoding spec).

Asn1Value attrlist;

Asn1Value opState;
Asn1Value usState;
Asn1Value sysTitle;
Asn1Value wInt;
Asn1Value rString;

Asn1Value sysIdO;
Asn1Value opStateO;
Asn1Value usStateO;
Asn1Value sysTitleO;
Asn1Value wIntO;
Asn1Value rStringO;

Asn1Value opStateV;
Asn1Value usStateV;
Asn1Value sysTitleV;
Asn1Value wIntV;
Asn1Value rStringV;

// Initialize the OIDs.

Oid sysIdOid("2.9.3.2.7.4"); // systemId
Oid opStateOid("2.9.3.2.7.35"); // operationalState
Oid usStateOid("2.9.3.2.7.39"); // usageState
Oid sysTitleOid("2.9.3.2.7.5"); // systemTitle
Oid wIntOid("1.2.3.4.5.6.7.10"); // writeableInteger
Oid rStringOid("1.2.3.4.5.6.7.11"); // readableString

```

CODE EXAMPLE 9-11 send_create_req (Continued)

```
// Initialize the five mandatory attributes
// in the mySystem class.

TTRYRES(opStateO.encode_oid(TAG_CONT(0), opStateOid));
TTRYRES(opStateV.encode_enum(TAG_ENUM, 1)); // enabled
TTRYRES(opState.start_construct(TAG_SEQ));
TTRYRES(opState.add_component(opStateO));
TTRYRES(opState.add_component(opStateV));

TTRYRES(usStateO.encode_oid(TAG_CONT(0), usStateOid));
TTRYRES(usStateV.encode_enum(TAG_ENUM, 1)); // active
TTRYRES(usState.start_construct(TAG_SEQ));
TTRYRES(usState.add_component(usStateO));
TTRYRES(usState.add_component(usStateV));

TTRYRES(sysTitleO.encode_oid(TAG_CONT(0), sysTitleOid));
TTRYRES(sysTitleV.encode_null(TAG_NULL));
TTRYRES(sysTitle.start_construct(TAG_SEQ));
TTRYRES(sysTitle.add_component(sysTitleO));
TTRYRES(sysTitle.add_component(sysTitleV));

TTRYRES(wIntO.encode_oid(TAG_CONT(0), wIntOid));
TTRYRES(wIntV.encode_int(TAG_INT, 5));
TTRYRES(wInt.start_construct(TAG_SEQ));
TTRYRES(wInt.add_component(wIntO));
TTRYRES(wInt.add_component(wIntV));

TTRYRES(rStringO.encode_oid(TAG_CONT(0), rStringOid));
TTRYRES(rStringV.encode_octets(TAG_GRAPHSTR,DataUnit
    ("test1")));
TTRYRES(rString.start_construct(TAG_SEQ));
TTRYRES(rString.add_component(rStringO));
TTRYRES(rString.add_component(rStringV));

// Create the attribute list.

TTRYRES(attrlist.start_construct(TAG_CONT(7))); //Implicit
    set of
TTRYRES(attrlist.add_component(opState));
TTRYRES(attrlist.add_component(usState));
TTRYRES(attrlist.add_component(sysTitle));
TTRYRES(attrlist.add_component(wInt));
TTRYRES(attrlist.add_component(rString));

// Send the confirmed create request. No user data is
// specified for the callback parameter. The create_req_cb
```

CODE EXAMPLE 9-11 `send_create_req` (Continued)

```

// function is included as part of the Object Services
// API examples.

oamsvc_test.print("About to issue Confirmed Create
Request\n");
if (send_action_req(mySystem_oc, mySystem_fdn, attrlist,
    Callback(create_req_cb, 0)) != OK)
    objsvc_test.print("Error issuing Confirmed Create
        Request\n");
    return NOT_OK;
}
else
    objsvc_test.print("Issued Confirmed Create Request\n");
}
VBEGHANDLERS
VCATCHALL {
objsvc_test.print("\nError encoding Create Request\n");
return NOT_OK;
}
VENDHANDLERS
return OK;
}

```

9.4.8 Create Response Callback

This function is used only in conjunction with confirmed create requests. The name of the callback function must match the name of the callback function specified in the callback parameter of the `send_create_req` function.

9.4.8.1 Interface Signature

```
void user-provided-create-response-callback(  
    Ptr userdata,           // Pointer to user supplied data  
    Ptr message);          // Pointer to CreateRes message
```

9.4.8.2 Create Response Callback Parameter Descriptions

TABLE 9-10 describes the Create Response Callback parameters.

TABLE 9-10 Create Response Callback Parameters

Parameter	Description
Ptr <i>userdata</i>	A void * pointer to data specified by the application developer in the callback parameter of the send_create_req function.
Ptr <i>message</i>	A void * pointer to the CreateRes message generated in response to the send_create_req function.

9.4.8.3 Create Response Callback Example

The mysystem.gdmo and mysystem.asn1 files must be loaded into the MDR for this example to work properly. The mysystem files can be found in /opt/SUNWconn/em/src/scenario/example1. The README file in this directory also describes how to load the mysystem files.

CODE EXAMPLE 9-12 Create Response Callback

```
void  
create_req_cb(Ptr, Ptr create_response_msg)  
{  
    objsvc_test.print("Create Request callback\n");  
    Message *resp = (Message *)create_response_msg;  
  
    VTRY {  
        if ( resp->type() == CREATE_RES) {  
            CreateRes *cr_resp = (CreateRes *)resp;  
            cr_resp->print(objsvc_test);  
  
            // The Asn1Value decoding functions, including the  
            // member functions get_first_component and  
            // get_next_component can be used to examine the member data
```


CODE EXAMPLE 9-12 Create Response Callback (*Continued*)

```

        // included in the response message at this point

        // ...

    }
    else if ( resp->is_error() ) {
        objsvc_test.print("Action response error received\n");
        objsvc_test.print( "message type = %s\n",
            MessageType_fmt(resp->type()));
        resp->print(objsvc_error);
    }
    else {
        objsvc_test.print(
            "Unexpected or invalid response received\n");
        objsvc_test.print( "message type = %s\n",
            MessageType_fmt(resp->type()));
        resp->print(objsvc_error);
    }
}
VBEGHANDLERS
VCATCHALL {
oamsvc_test.print("Error processing response for Action\n");
}
VENDHANDLERS

if ( resp )
    Message:: delete_message(resp);
}

```

9.4.9 Delete Request Service

`send_delete_req` enables you to override GDMO restrictions that are related to MO deletion operations.

This section includes the interface signature, `send_delete_req` parameters, and examples.

9.4.9.1 Delete Request Service

The rules for deleting an MO are defined in the MO class definition and the name bindings in a GDMO document. The rules which are defined in the name bindings impose restrictions on the deletion of MOs, but it is possible to override some of these restrictions when requesting deletion of an MO through the function `send_delete_req()`. Specifically, the absence of a `DELETE` construct in the name binding is intended to have the effect of restricting the deletion of an MO by a management application.

To allow behavior internal to the MIS to override this restriction, this function is defined with a `flags` parameter. The relevant flag value is `ReqMess::OVERRIDE_NAME_BINDING`, which is defined in `message.hh`.

Note that:

- The `ONLY-IF-NO-CONTAINED-OBJECTS` construct cannot be overridden.
- The MIS imposes a restriction on object deletion if an object contains other objects, irrespective of whether the `ONLY-IF-NO-CONTAINED-OBJECTS` construct appears in the name binding.
- If the `DELETES-CONTAINED-OBJECTS` construct appears in the name binding, the override capability only applies to the base object, and not to contained objects.
- If a delete request specifies a scope and a filter, the override capability only applies to the base object, and not to the scoped and filtered objects.

9.4.9.2 Interface Signature

CODE EXAMPLE 9-13 Delete Request Service: Interface Signature

```
Result send_delete_req(
    const Asn1Value &oc,
    const Asn1Value &oi,
    const Callback cb = 0, // Default: Unconfirmed
    MessId *id = 0,
```

CODE EXAMPLE 9-13 Delete Request Service: Interface Signature

```
Result send_delete_req(
    const MessScope scope = MessScope(),
        // Default: Base object only
    const Asn1Value filter = Asn1Value(),
        // Default: Matches all
    const MessSync sync = BEST_EFFORT,
    const Asn1Value access = Asn1Value(),
    Boolean sub_trans
); // Default: No access cntrl
```

9.4.9.3 send_delete_req Parameter Descriptions

TABLE 9-11 describes the `send_delete_req` parameters.

TABLE 9-11 `send_delete_req` Parameters

Parameter	Description	Required/ Optional
<code>const Asn1Value oc</code>	OID that specifies the class of the managed object.	Required
<code>const Asn1Value oi</code>	Distinguished name or local distinguished name for the managed object.	Required
<code>const Callback cb</code>	Specifies the name of a callback function invoked when a response to the delete request operation is received. Can also be used to specify user data passed to the callback function when it is invoked. The callback parameter is optional for the <code>send_delete_req</code> function. If specified, a confirmed delete request operation is issued and the callback function is invoked when the response is received. If not specified, an unconfirmed delete request operation is issued and no response is generated.	Optional
<code>MessId id</code>	Provides a unique identifier for a particular delete request operation. If specified, the value for this parameter is generated and set by the <code>send_delete_req</code> function.	Optional
<code>const MessScope scope</code>	Defines the type of scoping used for this request operation. If not specified, it defaults to <code>BASE_OBJECT_ONLY</code> .	Optional
<code>const Asn1Value filter</code>	Defines a filter to be passed by all objects selected by the scoping parameter. If not specified, it defaults to a filter that matches all managed objects selected by the scope parameter.	Optional
<code>const MessSync sync</code>	Defines the type of synchronization used for this request operation. If not specified, it defaults to <code>BEST_EFFORT</code> .	Optional
<code>Boolean sub_trans</code>	Spawn subtransactions instead of creating new transactions. If <code>sub_trans</code> is not specified, the default behavior is to create a new transaction for the operation.	Optional
<code>const Asn1Value access</code>	Reserved parameter. Do not use now.	Not available

9.4.9.4 *send_delete_req* Examples

CODE EXAMPLE 9-14 shows how to override restrictions on deleting an MO.

The `mysystem.gdmo` and `mysystem.asn1` files must be loaded into the MDR for this example to work properly. The `mysystem` files are located at `/opt/SUNWconn/em/src/scenario/example1`. The `README` file in this directory also describes how to load the `mysystem` files.

Delete Request: Base Object Only

CODE EXAMPLE 9-14 Delete Request: Base Object Only

```
Result
delete_mySystem_object(char *system_name, char *mySystem_name)
{
    Asn1Value mySystem_oc;
    Asn1Value mySystem_fdn;
    Asn1Value mySystem_rdn;

    //*****
    // Confirmed delete request
    //*****
    // Send a delete request to an instance of the
    // mySystem managed object class. The mySystem
    // instance to be deleted is contained by the system
    // object specified by system_name. The name of the
    // instance of the mySystem class to delete is specified
    // by the mySystem_name parameter.

    VTRY {

        // Encode oc OID. In this example, the OID for the
        // mySystem class is used.
        // Note: CMIP requires a TAG_CONT(0) encoding for the
        // object class rather than TAG_OID (refer to x711.asn1
        // for encoding spec).

        TTRYRES(mySystem_oc.encode_oid(
            TAG_CONT(0),Oid("1.2.3.4.5.6.3.10")));

        // Encode the distinguished name for an instance of the
        // mySystem managed object class. Instances of the mySystem
        // class are named using the systemId attribute in the class.
```

CODE EXAMPLE 9-14 Delete Request: Base Object Only *(Continued)*

```

// Either the distinguished name form (TAG_CONT(2))
// or the local distinguished name form (TAG_CONT(4)) can be
// used with the action operation. The distinguished name form
// is used in this example.
// The get_sys_dn and get_graphstr_rdn functions are included
// in Section 9.5. The definition for the sys_id (systemId)
OID
// is included in Section 9.5.

TTRYRES(get_sys_dn(system_name, mySystem_fdn));
TTRYRES(get_graphstr_rdn(sys_id, // systemId OID
    mySystem_name, mySystem_rdn));
TTRYRES(mySystem_fdn.add_component(mySystem_rdn));

// Send the confirmed Delete Request. No user data is
// specified for the callback parameter. The delete_req_cb
// function is included as part of the Object Services API
examples.

    objsvc_test.print("About to issue confirmed Delete
Request\n");
    if (send_delete_req(mySystem_oc, mySystem_fdn,
        Callback(delete_req_cb, 0)) != OK) {
        objsvc_test.print("Error issuing confirmed Delete
Request\n");
        return NOT_OK;
    }
    else
        objsvc_test.print("Issued confirmed Delete Request\n");
}
VBEGHANDLERS
VCATCHALL {
    objsvc_test.print("\nError encoding Delete Request\n");
    return NOT_OK;
}
VENDHANDLERS
return OK;
}

```

*Delete Request: Scoped Operation***CODE EXAMPLE 9-15** Delete Request: Scoped Operation

```

Result
delete_mySystem_object(char *system_name)
{
    Asn1Value actual_oc;
    Asn1Value system_fdn;

    //*****
    // Confirmed delete request
    //*****
    // Send a delete request to delete all instances
    // of the mySystem managed object class below a
    // system object. The system "base" object is
    // specified by the system_name parameter.
    // A CMIS filter is used to select only instances
    // of the mySystem class.

    VTRY {

        // Encode oc OID. In this example, actualClass is used
        // instead of the OID for the log managed object class.
        // Note: CMIP requires a TAG_CONT(0) encoding for the
        // object class rather than TAG_OID (refer to x711.asn1
        // for encoding spec).

        TTRYRES(actual_oc.encode_oid(TAG_CONT(0),
            Oid("2.9.3.4.3.42")));

        // Encode the distinguished name for the instance of the
        // system managed object class that is the base object for
        // the scoped operation.
        // Either the distinguished name form (TAG_CONT(2))
        // or the local distinguished name form (TAG_CONT(4)) can be
        // used with the delete operation. The distinguished name form
        // is used in this example.
        // The get_sys_dn and get_graphstr_rdn functions are included
        // in Section 9.5. The definition for the sys_id (systemId)
    OID
        // is included in Section 9.5.

        TTRYRES(get_sys_dn(system_name, system_fdn));

        // Encode the CMIS Filter. This filter checks for
        // "managedObjectClass == mySystem"

```

CODE EXAMPLE 9-15 Delete Request: Scoped Operation (*Continued*)

```

AsnlValue cmis_filter;
AsnlValue filter_item;
AsnlValue mOC_OID;
AsnlValue mySystem_oc_OID;

TTRYRES(cmis_filter.start_construct(TAG_CONT(8)));
TTRYRES(filter_item.start_construct(TAG_CONT(0)));
TTRYRES(mOC_OID.encode_oid(// ISO DMI
    TAG_CONT(0),Oid("2.9.3.2.7.60")));// managedObjectClass
TTRYRES(mySystem_oc_OID.encode_oid(
    TAG_CONT(0),Oid("1.2.3.4.5.6.3.10")));
TTRYRES(filter_item.add_component(mOC_OID));
TTRYRES(filter_item.add_component(mySystem_oc_OID));
TTRYRES(cmis_filter.add_component(filter_item));

// Send the confirmed scoped Delete Request. No user data is
// specified for the callback parameter. The delete_req_cb
// function is included as part of the Object Services API
examples.

    objsvc_test.print(
        "About to issue confirmed scoped Delete Request\n");
    if (send_delete_req(mySystem_oc, mySystem_fdn,
        Callback(scoped_del_req_cb, 0),
        MessScope(NTH_LEVEL, 1), cmis_filter) != OK) {
        objsvc_test.print(
            "Error issuing confirmed scoped Delete Request\n");
        return NOT_OK;
    }
    else
        objsvc_test.print("Issued confirmed scoped Delete
            Request\n");
    }
VBEGHANDLERS
VCATCHALL {
    objsvc_test.print("\nError encoding scoped Delete
        Request\n");
    return NOT_OK;
}
VENDHANDLERS
return OK;
}

```


9.4.10 Delete Response Callback

This callback function is used only in conjunction with the confirmed delete request service. The name of the callback function must match the name of the callback function specified in the callback parameter of the `send_delete_req` function.

9.4.10.1 Interface Signature

```
void user-provided-delete-response-callback(
    Ptr userdata,      // Pointer to user supplied data
    Ptr message);      // Pointer to DeleteRes message
```

9.4.11 Delete Response Callback Parameter Description

TABLE 9-12 describes the Delete Response Callback parameters.

TABLE 9-12 Delete Response Callback Parameters

Parameter	Description
Ptr <i>userdata</i>	A void * pointer to data specified by the application developer in the callback parameter of the <code>send_delete_req</code> function.
Ptr <i>message</i>	A void * pointer to the DeleteRes message generated in response to the <code>send_delete_req</code> function.

9.4.11.1 Delete Response Callback Examples

The `mySystem.gdmo` and `mySystem.asn1` files must be loaded into the MDR for these examples to work properly. The `mySystem` files are located at `/opt/SUNWconn/em/src/scenario/example1`. The README file in this directory also describes how to load the `mySystem` files.

*Delete Response Callback Function: Single Response (Base Object Only)***CODE EXAMPLE 9-16** Delete Response Callback Function: Single Response

```

void
del_req_cb(Ptr, Ptr delete_response_msg)
{
    objsvc_test.print("Get Request callback\n");
    Message *resp = (Message *)delete_response_msg;

    VTRY {
        if ( resp->type() == DELETE_RES) {
            DeleteRes *d_resp = (DeleteRes *)resp;
            d_resp->print(objsvc_test);

            // The Asn1Value decoding functions, including the
            // member functions get_first_component and
            // get_next_component can be used to examine the member data
            // included in the response message at this point

            // ...

        }
        else if ( resp->is_error() ) {
            objsvc_test.print("Delete response error received\n");
            objsvc_test.print( "message type = %s\n",
                               MessageType_fmt(resp->type()));
            resp->print(objsvc_error);
        }
        else {
            objsvc_test.print(
                "Unexpected or invalid response received\n");
            objsvc_test.print( "message type = %s\n",
                               MessageType_fmt(resp->type()));
            resp->print(objsvc_error);
        }
    }
    VBEGHANDLERS
    VCATCHALL {
        objsvc_test.print("Error processing response for Delete\n");
        resp->print(objsvc_error);
    }
    VENDHANDLERS

    if ( resp )
        Message::delete_message(resp);
}

```

Delete Response Callback Function: Multiple Responses (Scoped Operation)

CODE EXAMPLE 9-17 Delete Response Callback Function: Multiple Responses

```
static int d_resp_count = 0;
static int d_err_count = 0;
static int d_unknown_count = 0;
void
scoped_del_req_cb(Ptr userdata, Ptr delete_response_msg)
{
    objsvc_test.print("Scoped Delete Request callback:");
    Message *resp = (Message *)delete_response_msg;

    VTRY {
        if ( resp->type() == DELETE_RES ) {
            DeleteRes *sd_resp = (DeleteRes *)resp;
            if (sd_resp->linked) {
                objsvc_test.print("*** LINKED Response ***\n");
                resp_count++;
                sd_resp->print(objsvc_test);

                // The Asn1Value decoding functions, including the
                // member functions get_first_component and
                // get_next_component can be used to examine the member
                // data included in the response message at this point

                // ...

            } else {
                objsvc_test.print("**** Final Response ****\n");
                sd_resp->print(objsvc_test);
                objsvc_test.print("Valid: %d, Error: %d, Invalid:
                %d\n",
                resp_count, err_count, unknown_count);

                // Final response processing can be performed here. The
                // final response message does not contain any attribute
                // value data.

                // ...

            }
        }
    }
    else if ( resp->is_error() ) {
        objsvc_test.print("Error Response\n");
        objsvc_test.print( "message type = %s\n",
```

CODE EXAMPLE 9-17 Delete Response Callback Function: Multiple Responses *(Continued)*

```
        MesstType_fmt(resp->type());
resp->print(objsvc_error);
err_count++;
    }
    else {
        objsvc_test.print("Invalid Message\n");
        objsvc_test.print( "message type = %s\n",
            MesstType_fmt(resp->type()));
        resp->print(objsvc_error);
        unknown_count++;
    }
}
}
VBEGHANDLERS
VCATCHALL {
    objsvc_test.print(
        "\nError processing response for Scoped Delete\n");
}
VENDHANDLERS

if ( resp )
    Message:: delete_message(resp);
}
```

9.4.12 Event Report Request Service (Unconfirmed)

This section includes the interface signature, `send_event_req` parameters, and examples.

9.4.12.1 Interface Signature

CODE EXAMPLE 9-18 Event Report Request: Interface Signature

```
Result send_event_req(
    const Asn1Value &oc,
    const Asn1Value &oi,
    const Asn1Value &event_type,
    const Asn1Value event_info = ACEGlobals::null_asn1_value,
                                   // Default: No event info
    const Asn1Value event_time = ACEGlobals::null_asn1_value,
                                   // Default: No value
    const Callback cb = null_callback, // Default: Unconfirmed
    MessId *id = 0);
```

Note – The `send_event_req` service supports only unconfirmed operations. Specifying a callback generates a runtime error.

9.4.12.2 send_event_req Parameter Descriptions

TABLE 9-13 describes the `send_event_req` parameters.

TABLE 9-13 send_event_req Parameters

Parameter	Description	Required/ Optional
const Asn1Value <i>oc</i>	OID that specifies the class of the base managed object.	Required
const Asn1Value <i>oi</i>	Distinguished name or local distinguished name for the base managed object.	Required
const Asn1Value <i>event_type</i>	OID that specifies the type of notification generated by the <code>send_event_req</code> function.	Required

TABLE 9-13 `send_event_req` Parameters (*Continued*)

Parameter	Description	Required/ Optional
<code>const Asn1Value event_info</code>	Specifies any event information associated with the type of notification generated. This is an optional parameter but must be present if a <code>WITH INFORMATION SYNTAX</code> construct is specified as part of the GDMO definition for the notification type.	Optional
<code>const Asn1Value event_time</code>	The time at which the notification is generated.	Optional
<code>const Callback cb</code>	An optional reserved parameter for the event report request operation and should not be specified. Only unconfirmed (no callback function) event request operations are currently supported using this function call. If a value for this parameter is specified (in order to specify an <code>id</code> parameter, the <code>cb</code> parameter needs to be specified), it must either be set to <code>Callback()</code> or <code>Callback(0,0)</code> . Any other value generates an error when the <code>send_event_req</code> function is invoked.	Optional reserved
<code>MessId id</code>	Message identifier that uniquely identifies this request operation. If specified, the value for this parameter is generated and set by the <code>send_event_req</code> function.	Optional

9.4.12.3 `send_event_req` Example

The GDMO and definition for the notification generated by this example is as follows:

CODE EXAMPLE 9-19 `send_event_req`

```
connectivityChange NOTIFICATION
    BEHAVIOUR connectivityChangeBehaviour BEHAVIOUR DEFINED AS
        !Generated by an instance of the connectMgr object
        when the state of a connection between two MISS
        changes.!!
    ;
    WITH INFORMATION SYNTAX ConnectMgr-
    ASN1.ConnectivityChangeInfo
```

CODE EXAMPLE 9-19 send_event_req (*Continued*)

```
REGISTERED AS { connectivityMgmt 10 1 };
```

The corresponding ASN.1 definitions are as follows:

```
ConnectMgr-ASN1 { 6 2 3 4 5 6 7 2 1 }
```

```
...
```

```
ConnectState ::= ENUMERATED
```

```
{
    notConnected(0),
    connected(1),
    errorDisconnect(2),
    resyncDisconnect(3),
    badapplicationId(4)
}
```

```
...
```

```
ConnectivityChangeDefinition ::= SEQUENCE
```

```
{
    remoteMIS          SystemId,
    previousState      ConnectState,
    currentState       ConnectState
}
```

```
ConnectivityChangeInfo ::= ConnectivityChangeDefinition
```

Example code to generate notification using send_event_req function is:

```
Result
```

```
send_conn_change_event(char *from_sys, char *remote_sys)
```

```
{
    Asn1Value conn_oc;
    Asn1Value conn_oi;
    Asn1Value em_mis_rdn;
    Asn1Value conn_rdn;
    Asn1Value conn_evt_type;
    Asn1Value conn_evt_info;
    Asn1Value rmt_id;
    Asn1Value rmt_prev;
    Asn1Value rmt_curr;

    //*****
    // Unconfirmed event report request
    //*****
    // Issue a connectivity change notification
    // This notification is defined by the connection
    // manager object (refer to connmgr.gdmo/connmgr.asn1)
    // and is issued when the connectivity state between
    // two MISs changes. In this example, the two MISs
    // are from_sys (which generates the notification) and
```

CODE EXAMPLE 9-19 `send_event_req` (Continued)

```

// remote_sys.

VTRY {

// Encode oc OID for connection manager object class
// which is defined in connmgr.gdmo. Note: CMIP requires
// a TAG_CONT(0) encoding for the object class rather
// than TAG_OID (refer to x711.asn1 for encoding spec).

TTRYRES(conn_oc.encode_oid(
    TAG_CONT(0),Oid("1.3.6.1.4.1.42.2.2.2.201.3.1")));

// Encode the distinguished name for the connection manager
// managed object that generates the notification.
// The send_event_req function requires the distinguished
// name form (TAG_CONT(2)) for the object instance.

TTRYRES(get_sys_dn(from_sys, conn_oi));
TTRYRES(get_graphstr_rdn(
    "2.9.3.5.7.11","EM-MIS", em_mis_rdn));
TTRYRES(get_graphstr_rdn(
    "1.3.6.1.4.1.42.2.2.2.201.7.3","ConnectMgr",
    conn_rdn));
TTRYRES(conn_oi.add_component(em_mis_rdn));
TTRYRES(conn_oi.add_component(conn_rdn));

// Encode the connectivityChange event type and event info
// The connectivityChange OID is defined in connmgr.gdmo.
// The format of the corresponding event info is defined
// in connmgr.asn1.
// Note: CMIP requires a TAG_CONT(6) for the event type OID
// rather than a TAG_OID and also requires a TAG_CONT(8)
// encoding for the event info rather than a TAG_SEQ.
TTRYRES(conn_evt_type.encode_oid(
    TAG_CONT(6),Oid("1.3.6.1.4.1.42.2.2.2.201.10.1")));
TTRYRES(conn_evt_info.start_construct(TAG_CONT(8)));
TTRYRES(rmt_id.encode_octets(
    TAG_GRAPHSTR, DataUnit(remote_sys)));
TTRYRES(rmt_prev.encode_enum(TAG_ENUM, 0)); // notConnected
TTRYRES(rmt_curr.encode_enum(TAG_ENUM, 1)); // connected
TTRYRES(conn_evt_info.add_component(rmt_id));
TTRYRES(conn_evt_info.add_component(rmt_prev));
TTRYRES(conn_evt_info.add_component(rmt_curr));

// Send the Event Report Request

```


CODE EXAMPLE 9-19 send_event_req (Continued)

```

objsvc_test.print(
    "About to issue Unconfirmed EVENT REPORT Request\n");
if (send_event_req(conn_oc, conn_oi, conn_evt_type,
    conn_evt_info) != OK) {
    objsvc_test.print(
        "Error issuing Unconfirmed EVENT REPORT Request\n");
    return NOT_OK;
}
else
    objsvc_test.print("Issued Unconfirmed EVENT REPORT
        Request\n");
}
VBEGHANDLERS
VCATCHALL {
    objsvc_test.print("\nError encoding EVENT REPORT
        Request\n");
    return NOT_OK;
}
VENDHANDLERS
return OK;
}

```

9.4.13 Event Report Response Callback

This callback function is not supported for the Solstice EM services interface.

9.5 Supporting Functions for Example Code

This section includes the following information:

- Debugging flags
- get_sys_dn function
- get_graphstr_rdn function

9.5.1 Debugging Flags

The following code lines must be included in the file that contains the example functions.

```
Debug_on(objsvc_test)
Debug_on(objsvc_error)
// Note: Do not include a semicolon after these two lines.
```

If the example functions are spread across multiple files, the above definitions must only be included in one file. The other files need to contain the following code lines:

```
extern Debug objsvc_test;
extern Debug objsvc_error;
```

The debug flags can be enabled using the following commands:

```
/opt/SUNWconn/em/bin/em_debug "on objsvc_test"
/opt/SUNWconn/em/bin/em_debug "on objsvc_error"
```

or alternatively using the following command:

```
/opt/SUNWconn/em/bin/em_debug "on objsvc_*
```

9.5.2 get_sys_dn Function

The `get_sys_dn` function encodes a distinguished name for an instance of the system managed object class.

CODE EXAMPLE 9-20 `get_sys_dn` Function

```
// Function to encode a distinguished name (TAG_CONT(2) for
// an instance of the system managed object class. The encoding
// assumes that the instance is contained directly under root
Oid      sys_id((char *)"2.9.3.2.7.4"); // ISO DMI systemId OID

Result
get_sys_dn(const char *sys_nm, Asn1Value &sys_fdn)
{
    Asn1Value sys_rdn;
```

CODE EXAMPLE 9-20 get_sys_dn Function (Continued)

```

// Function to encode a distinguished name (TAG_CONT(2) for
    Asn1Value sys_ava;
    Asn1Value sys_name;
    Asn1Value sys_oid;

    if (!sys_nm)
        return NOT_OK;

    VTRY{
        TTRYRES(sys_fdn.start_construct(TAG_CONT(2)));
        TTRYRES(sys_rdn.start_construct(TAG_SET));
        TTRYRES(sys_ava.start_construct(TAG_SEQ));
        TTRYRES(sys_oid.encode_oid(TAG_OID, sys_id));
        TTRYRES(sys_name.encode_octets(TAG_GRAPHSTR, sys_nm));
        TTRYRES(sys_ava.add_component(sys_oid));
        TTRYRES(sys_ava.add_component(sys_name));
        TTRYRES(sys_rdn.add_component(sys_ava));
        TTRYRES(sys_fdn.add_component(sys_rdn));
    }
    VBEGHANDLERS
    VCATCHALL
        objsvc_test.print("get_sys_dn: error encoding DN\n");
        return NOT_OK;
    VENDHANDLERS

    return OK;
}

```

9.5.3 get_graphstr_rdn Function

The get_graphstr_rdn function are described in CODE EXAMPLE 9-21.

CODE EXAMPLE 9-21 get_graphstr_rdn Function

```

// Function to encode an RDN consisting of an
// object identifier (OID) and an ASN.1 GraphicString
Result
get_graphstr_rdn(const char *a_oidstr, const char *a_str,
    Asn1Value &a_rdn)
{
    Asn1Value a_ava;
    Asn1Value a_oid;
    Asn1Value a_val;

    if (!a_oidstr || !a_str)

```

CODE EXAMPLE 9-21 `get_graphstr_rdn` Function (Continued)

```

        return NOT_OK;

    TRY {
        TTRYRES(a_rdn.start_construct(TAG_SET));
        TTRYRES(a_ava.start_construct(TAG_SEQ));
        TTRYRES(a_oid.encode_oid(TAG_OID, Oid(a_oidstr)));
        TTRYRES(a_val.encode_octets(TAG_GRAPHSTR,
            DataUnit(a_str)));
        TTRYRES(a_ava.add_component(a_oid));
        TTRYRES(a_ava.add_component(a_val));
        TTRYRES(a_rdn.add_component(a_ava));
    }
    VBEGHANDLERS
    VCATCHALL {
        objsvc_test.print("\nError encoding RDN for %s\n", a_val);
        return NOT_OK;
    }
    VENDHANDLERS

    return OK;
}

// Function to encode an RDN consisting of an object identifier
// (OID)
// and an ASN.1 GraphicString
Result
get_graphstr_rdn(const Oid &a_oidval, const char *a_str,
Asn1Value
    &a_rdn)
{
    Asn1Value a_ava;
    Asn1Value a_oid;
    Asn1Value a_val;

    if (!a_str)
        return NOT_OK;

    TRY {
        TTRYRES(a_rdn.start_construct(TAG_SET));
        TTRYRES(a_ava.start_construct(TAG_SEQ));
        TTRYRES(a_oid.encode_oid(TAG_OID, a_oidval));
        TTRYRES(a_val.encode_octets(TAG_GRAPHSTR,
            DataUnit(a_str)));
        TTRYRES(a_ava.add_component(a_oid));
        TTRYRES(a_ava.add_component(a_val));
        TTRYRES(a_rdn.add_component(a_ava));
    }

```

CODE EXAMPLE 9-21 `get_graphstr_rdn` Function (*Continued*)

```
VBEGHANDLERS
VCATCHALL {
    objsvc_test.print("\nError encoding RDN for %s\n", a_val);
    return NOT_OK;
}
VENDHANDLERS

    return OK;
}
```


Index

A

ACAccessControlRules class, 5-12

attributes

accessControlSwitch, 5-12

trustedHostList, 5-12

constructor, 5-12

description, 5-3

destructor, 5-13

member functions

add_trusted_hosts, 5-13

get_access_control_switch, 5-13

get_default_access, 5-14

get_default_event_access, 5-14

get_denial_granularity, 5-15

get_denial_response, 5-15

get_domain_identity, 5-16

get_trusted_host_list, 5-16

is_trusted_host, 5-16

remove_trusted_hosts, 5-16

replace_trusted_host_list, 5-17

set_access_control_switch, 5-17

set_default_access, 5-17

set_default_event_access, 5-18

set_denial_granularity, 5-18

set_denial_response, 5-19

ACAccessUserList class, 5-20

constructor, 5-20

description, 5-3

destructor, 5-20

member functions

add_superuser, 5-21

add_user, 5-21

get_access_user_list_set, 5-21

get_superuser_list, 5-21

is_superuser, 5-22

remove_superusers, 5-22

remove_user, 5-22

replace_superuser_list, 5-22

ACAppFeatureContainer class, 5-23

constructor, 5-23

description, 5-3

destructor, 5-23

member functions

get_all_features, 5-23

get_container_name, 5-24

get_feature, 5-24

ACApplication class, 5-24

constructor, 5-24

description, 5-3

member functions

destroy, 5-25

get_application_description, 5-25

ACApplicationContainer class, 5-26

constructor, 5-26

description, 5-3

destructor, 5-26

member functions

get_all_applications, 5-26

get_application, 5-27

ACApplicationFeature class, 5-27

constructor, 5-27

description, 5-3

destructor, 5-28

member functions

destroy, 5-28

get_feautre_description, 5-28

set_feature_description, 5-28

- ACCallback class, 5-29
 - constructors, 5-29
 - description, 5-3
 - destructor, 5-30
 - member functions
 - exec_callback, 5-30
 - get_callback_type, 5-30
 - operator overloading, 5-30
- access control
 - types
 - feature-level, 5-2
 - object-level, 5-2
- access control API, 5-1
 - class hierarchy, 5-2
 - constants
 - ACAccessControlSwitch, 5-5
 - ACAuxOwnerType, 5-4
 - ACCallbackType, 5-5
 - ACDenialGranularity, 5-5
 - ACEMAuditLevel, 5-5
 - ACEMSecurityLevel, 5-6
 - ACErrorType, 5-6
 - ACObjectType, 5-7
 - ACTargetsType, 5-7
 - EnforcementAction, 5-7
 - defined types
 - ACAccessUserListSet, 5-8
 - ACApplicationAndFeatureList, 5-8
 - ACApplicationFeatureList, 5-8
 - ACDomainIdentity, 5-9
 - ACEventsDiscriminator, 5-9
 - ACFilter, 5-9
 - ACGroupDescription, 5-10
 - ACGroupList, 5-10
 - ACGroupMemberList, 5-10
 - ACMOCList, 5-10
 - ACMOIList, 5-10
 - ACOperationsList, 5-10
 - ACRuleList, 5-10
 - ACSuperUserList, 5-11
 - ACTargetsList, 5-11
 - ACTrustedHostList, 5-11
 - design objectives, 5-1
- access control API classes, list of, 5-11
- access control engine API
 - symbolic constants
 - ACEEnforcementAction, 6-2
 - ACEOperationType, 6-2
- AccessDenied class, 4-9
- ACContainer class, 5-31
 - constructor, 5-31
 - description, 5-3
 - destructor, 5-31
 - member functions
 - add_callback, 5-32
 - get_error_string, 5-33
 - get_error_type, 5-33
 - get_name_only, 5-33
 - get_object_name, 5-34
 - remove_callback, 5-34
 - reset_error, 5-34
 - set_error, 5-34
 - set_error_string, 5-34
 - set_error_type, 5-35
 - operator overloading, 5-32
- ACDBObject class, 5-35, 5-38
 - constructor, 5-35
 - description, 5-4
 - destructor, 5-35
 - member functions
 - add_db_object_access, 5-36
 - add_db_object_table, 5-36
 - det_db_object_access_list, 5-36
 - get_auxobject_owner_id, 5-38
 - get_auxobject_owner_type, 5-38
 - get_db_object_table_list, 5-36
 - remove_db_object_access, 5-37
 - remove_db_object_table, 5-37
 - set_auxobject_owner, 5-38
 - set_db_object_access_list, 5-37
 - set_db_object_table_list, 5-37
- ACDBObjectContainer class, 5-40
 - constructor, 5-40
 - description, 5-4
 - destructor, 5-40
 - member functions
 - get_access_db_objects, 5-40
 - get_all_db_objects, 5-41
 - get_db_object, 5-41
 - get_db_server_name, 5-41
 - get_db_server_type, 5-41
- ACE API (Access Control Engine API)
 - uses, 6-1
- ACE API classes, list of, 6-3
- ACE class, 6-3
 - constructor, 6-4
 - destructor, 6-4
 - member functions

- check_access, 6-4
 - get_ace_instance function, 6-4
 - hi_process_ace_event, 6-6
 - lo_process_ace_event, 6-6
- ACEContext class, 6-6
 - constructor, 6-7
 - destructor, 6-7
 - member functions
 - get_filter, 6-8
 - get_orig_user_req, 6-7
 - get_scope, 6-8
 - set_filter, 6-8
 - set_scope, 6-8
- ACEDecision class, 6-9
 - constructor, 6-9
 - destructor, 6-10
- ACEDomain class, 6-11
 - constructor, 6-11
 - destructor, 6-11
- ACEMNotificationEmitter class, 5-42
 - constructor, 5-42
 - description, 5-4
 - destructor, 5-42
 - member functions
 - get_audit_level, 5-43
 - get_invalid_access_attempts, 5-43
 - get_security_level, 5-43
 - get_valid_access_attempts, 5-43
 - set_audit_level, 5-44
 - set_security_level, 5-44
- ACEMTargets class, 5-44
 - constructor, 5-45
 - description, 5-4
 - destructor, 5-45
 - member functions
 - get_event_discriminator, 5-45
 - set_event_discriminator, 5-45
- ACEResData class, 6-12
 - destructor, 6-12, 6-13
- ACGroup Class
 - member functions
 - add_application_feature, 5-48
- ACGroup class, 5-46
 - constructor, 5-46
 - description, 5-4
 - destructor, 5-46
 - member functions
 - add_application, 5-47
 - add_application_feature, 5-47
 - add_group_member, 5-47
 - destroy, 5-47
 - get_all_applications_full_access, 5-48
 - get_application_and_feature_list, 5-48
 - get_application_features, 5-49
 - get_application_full_access, 5-49
 - get_applications, 5-48
 - get_group_description, 5-49
 - get_group_member_list, 5-49
 - get_initiator_aci_mandated, 5-52
 - remove_application, 5-50
 - remove_application_feature, 5-50
 - remove_group_member, 5-50
 - set_all_applications_full_access, 5-50
 - set_application_and_feature_list, 5-51
 - set_application_full_access, 5-51
 - set_group_description, 5-51
 - set_group_member_list, 5-52
 - set_initiator_aci_mandated, 5-52
- ACGroupContainer class, 5-52
 - constructor, 5-53
 - description, 5-4
 - destructor, 5-53
 - member functions
 - get_all_groups, 5-53
 - get_group, 5-53
 - get_user_group_list, 5-54
- ACInterface class, 5-54
 - constructor, 5-54
 - description, 5-4
 - destructor, 5-55
 - member functions
 - get_access_user_list, 5-55
 - get_application_container, 5-55
 - get_db_object_container, 5-55
 - get_em_notification_emitter, 5-55
 - get_feature_container, 5-56
 - get_group_container, 5-56
 - get_rule_container, 5-56
 - get_targets_container, 5-56
- ACObject class, 5-57
 - description, 5-4
 - destructor, 5-57
 - member functions
 - add_callback, 5-58

- copy, 5-59
- create, 5-59
- destroy, 5-59
- exists, 5-59
- get_error_string, 5-60
- get_error_type, 5-60
- get_name_only, 5-60
- get_object_name, 5-61
- remove_callback, 5-61
- revert, 5-61
- set_error, 5-61
- set_error_string, 5-62
- set_error_type, 5-62
- store, 5-62
- operator overloading, 5-58
- ACObjectcClass
 - constructor, 5-57
- ACRule Class, 5-62
- ACRule class
 - constructor, 5-63
 - description, 5-4
 - destructor, 5-63
 - member functions
 - add_group, 5-63
 - add_targets, 5-64
 - get_enforcement_action, 5-64
 - get_group_list, 5-64
 - get_target_list, 5-64
 - remove_group, 5-65
 - remove_targets, 5-65
 - set_enforcement_action, 5-65
 - set_group_list, 5-66
 - set_targets_list, 5-66
- ACRuleContainer class, 5-66
 - constructor, 5-66
 - description, 5-4
 - destructor, 5-67
 - member functions
 - get_access_control_rules, 5-67
 - get_all_rules, 5-67
 - get_group_rule_list, 5-67
 - get_rule, 5-67
 - get_targets_rule_list, 5-68
- ACScope class, 5-68
 - constructors, 5-68
 - description, 5-4
 - operator overloading, 5-69
- ACTargets class, 5-69
 - constructor, 5-70

- description, 5-4
- destructor, 5-70
- member functions
 - add_moc, 5-70
 - add_moi, 5-71
 - destroy, 5-71
 - get_filter, 5-71
 - get_moc_list, 5-71
 - get_moi_list, 5-72
 - get_operations_list, 5-72
 - get_scope, 5-72
 - remove_moc, 5-72
 - remove_moi, 5-73
 - set_filter, 5-73
 - set_moc_list, 5-73
 - set_moi_list, 5-73
 - set_operations_list, 5-74
 - set_scope, 5-74
- ACTargetsContainer class, 5-74
 - constructor, 5-74
 - description, 5-4
 - destructor, 5-75
 - member functions
 - get_all_targets, 5-75
 - get_em_targets, 5-75
 - get_targets, 5-75
- ActionReq class, 4-10
- ActionRes class, 4-11
- ACUser class, 5-76
 - constructors, 5-76
 - description, 5-4
 - member functions
 - get_full_name, 5-77
 - get_login_name, 5-77
 - is_valid_user, 5-78
 - set_full_name, 5-78
 - set_login_name, 5-78
- Address class, 2-5
 - address tag, 2-5
 - address value, 2-5
 - public variables, 2-6
- ADF (Access Control Decision Function) operation, 6-1
- AEF (Access Control Enforcement) operation, 6-1
- Album class, 3-27
 - constructors
 - Album, 3-29
 - copy, 3-29
 - default, 3-29
 - events

- IMAGE_EXCLUDED, 3-52
- IMAGE_INCLUDED, 3-52
- OBJECT_CREATED, 3-52
- OBJECT_DESTROYED, 3-52
- RAW_EVENT, 3-52
- filtering derivation, 3-7
- member functions
 - all, 3-30
 - all_boot, 3-31
 - all_call, 3-32
 - all_create, 3-32
 - all_create_within, 3-33
 - all_destroy, 3-33
 - all_revert, 3-34
 - all_set, 3-34
 - all_set_attr_prop, 3-34
 - all_set_dbl, 3-35
 - all_set_from_ref, 3-35
 - all_set_gint, 3-35
 - all_set_long, 3-35
 - all_set_prop, 3-36
 - all_set_raw, 3-36
 - all_set_str, 3-36
 - all_shutdown, 3-37
 - all_start, 3-37
 - all_start_boot, 3-37
 - all_start_create, 3-38
 - all_start_create_within, 3-38
 - all_start_destroy, 3-38
 - all_start_raw, 3-39
 - all_start_shutdown, 3-39
 - all_start_store, 3-39
 - all_store, 3-39
 - all_when, 3-40
 - clear, 2-83, 3-40
 - derive, 3-40
 - destroy, 2-83
 - example, 3-47
 - exclude, 3-41
 - fetch, 2-84
 - find_by_nickname, 3-41
 - first_image, 3-41
 - get_derivation, 3-42
 - get_prop, 3-42
 - get_userdata, 3-43
 - get_when_syntax, 3-43
 - include, 3-43
 - iterate, 2-84
 - num_images, 3-44
 - set_derivation, 3-44
 - set_prop, 3-44
 - set_userdata, 3-44
 - start_m_action, 3-46
 - start_m_action_raw, 3-46
 - start_m_delete, 3-46
 - start_m_get, 3-45
 - start_m_set, 3-45
 - statr_derive, 3-45
 - strhash, 2-85
 - when, 3-51
- method types, 3-27
- operators
 - assignment operator, 3-30
 - cast operator, 3-30
 - not operator, 3-30
- properties, 3-4
 - ACCESS, 3-43
 - AUTOIMAGE, 3-43
 - BEST_EFFORT, 3-43
 - DERIVATION, 3-42
 - NICKNAME, 3-42
 - STATE, 3-42
 - TRACKMODE, 3-42
- properties supported, 3-42
- album synchronization, 3-27
- AlbumImage class, 3-52
 - constructors, 3-53
 - member functions
 - next_album, 3-55
 - next_image, 3-55
 - operators
 - Album operator, 3-55
 - assignment operator, 3-54
 - cast operator, 3-54
 - Image, 3-55
 - Not operator, 3-54
- AppEventHandler
 - set_indication_handler(), 1-10
- AppInstComm class, 1-21
 - member functions, 1-21
 - build_target, 1-22
 - DataFormatter, 1-22
 - send_request, 1-24
 - send_request_unconfirmed, 1-23
 - set_indication_handler, 1-25
 - start_send_request, 1-24
- AppInstObj class, 1-26
 - constructors, 1-26

- member functions
 - get_objname, 1-27
 - get_oi, 1-27
- AppRequest class, 1-28
 - emSendApplicationMessage action, 1-29
 - member functions
 - begin, 1-28
 - get_action, 1-29
 - get_receiver, 1-29
 - get_reply_data, 1-28
 - is_complete, 1-29
 - notifications, 1-30
- AppTarget class, 1-35, 3-56
- Arraydeclare macro, 2-8
- ASN.1 textual data, 3-3
- Asn1Kind declaration, 2-96
- Asn1ParsedValue class, 2-9
 - member functions
 - format_value, 2-10
 - get_parsed_val, 2-10
 - get_real_val, 2-11
 - operators, 2-10
- Asn1SubTypeKind declaration, 2-95
- Asn1SubTypeSize declaration, 2-95
- Asn1Tag class, 2-11
 - constructors, 2-12
 - member functions, 2-13
 - operator overloading, 2-12
 - public functions, 2-12
 - public variables, 2-12
- Asn1TagClass declaration, 2-97
- Asn1Tagging declaration, 2-97
- Asn1Type class, 2-13
 - code example
 - get_range(), 2-22
 - constructors, 2-15
 - destructor, 2-16
 - member functions
 - add_tags, 2-17
 - base_kind, 2-17
 - base_type, 2-18
 - cmp, 2-18
 - determine_real_val, 2-18
 - encode, 2-18
 - equivalent, 2-19
 - find_component, 2-19
 - find_subcomponent, 2-20
 - format_type, 2-20
 - format_value, 2-20
 - get_enum_identifiers, 2-21
 - get_range, 2-21
 - kind, 2-22
 - lookup_type, 2-23, 2-24, 2-26
 - needs_explicit, 2-24
 - parse_value, 2-24
 - register_any_handler, 2-24
 - remove_tags, 2-24
 - set_add_members, 2-25
 - set_intersects_with, 2-25
 - set_is_subset, 2-25
 - set_remove_dup_members, 2-26
 - set_remove_members, 2-26
 - unregister_any_handler, 2-26
 - validate, 2-27
 - validate_tag, 2-27
 - operator overloading, 2-16
 - public functions, 2-14
- Asn1TypeDefinedType declarations, 2-94
- Asn1TypeE declaration, 2-96
- Asn1TypeEL declaration, 2-96
- Asn1TypeNN declaration, 2-97
- Asn1Value class, 2-27
 - constructors, 2-31
 - decoding constructed Asn1Values, 2-30
 - decoding functions, 2-30
 - decoding simple Asn1Values, 2-30
 - delete operator, 2-28
 - destructor, 2-32
 - encoding a distinguished name, 2-29
 - encoding functions, 2-28
 - global functions, related, 2-48
 - instance assignment, 2-28
 - member functions
 - add_component, 2-34
 - compute_total_size, 2-34
 - constructed, 2-34
 - contents_size, 2-35
 - decode_bits, 2-35
 - decode_boolean, 2-35
 - decode_enum, 2-36
 - decode_ext, 2-36
 - decode_int, 2-37
 - decode_octets, 2-37
 - decode_oid, 2-37
 - decode_real, 2-38
 - decode_unsigned, 2-38
 - delete_component, 2-38
 - encode_boolean, 2-39

- encode_enum, 2-40
- encode_ext, 2-40
- encode_int, 2-41
- encode_minus_infinity, 2-41
- encode_null, 2-42
- encode_octets, 2-42
- encode_oid, 2-43
- encode_oidstr, 2-43
- encode_plus_infinity, 2-43
- encode_real, 2-44
- encode_unsigned, 2-44
- first_component, 2-44
- get_component, 2-45
- indefinite_length, 2-45
- make_explicit_tagged, 2-45
- next_component, 2-46
- num_comps, 2-46
- print, 2-46
- retag, 2-47
- size, 2-47
- start_construct, 2-47
- Tag, 2-47
- tagged_component, 2-48

- new operator, 2-28, 2-31
- operator overloading, 2-28, 2-33
- type conversion, 2-28

AssocReleased class, 4-12

AuthApps class, 3-56

AuthFeatures class, 3-58

AuthPriv class, 3-56

Auxiliary Servers, 6-1

AuxServerUtils class, 6-14

- constructor, 6-14

- destructor, 6-14

- virtual functions

- aux_check_create_filter, 6-16

- aux_check_event_filter, 6-17

- aux_get_red, 6-16

- check_filter, 6-15

- determine_class, 6-15

- extract_message, 6-16

AVData class, 2-28

AVData instance, 2-28, 2-31, 2-32, 2-33

B

basic variable types, 2-3

basic variables, types of, 2-3

BER (Basic Encoding Rules), 2-28

BER encoding, 2-28

Blockage class, 2-49

- global functions, related

- flush_events_callbacks, 2-54

- post_callback, 2-52

- post_fd_except_callback, 2-53

- post_fd_read_callback, 2-53

- post_fd_write_callback, 2-53

- purge_callback, 2-52

- purge_callback_cdata, 2-52

- purge_callback_data, 2-52

- purge_callback_handler, 2-53

- purge_fd_callbacks, 2-54

- purge_fd_except_callback, 2-54

- purge_fd_read_callback, 2-54

- purge_fd_write_callback, 2-54

- member functions

- purge_call, 2-50

- size, 2-50

- sleep, 2-50

- wakeup, 2-51

- wakeup_call, 2-51

- wakeup_now, 2-51

C

caching, 3-1

Callback class, 2-55

CancelGetReq class, 4-13

CancelGetRes class, 4-14

class destructors, 3-26

- example, 3-26

ClassInstConfl class, 4-15

client, notifications, 8-91

CmipAgent

- attributes, 8-112

CMIS, 3-2

CMIS message type

- error response, 4-4

- request, 4-4

- response, 4-4

CMIS-like protocol, 4-1

Coder class, 3-60

- decoding, 3-60

- encoding, 3-60

CoderData class, 3-60

common API class, 2-1

- categories, 2-2
- descriptions, 2-4
- CreateReq class, 4-16
- CreateRes class, 4-17
- CurrentEvent class, 3-62
- constructors
 - calldata constructor, 3-64
 - copy constructor, 3-64
 - default constructor, 3-64
- member functions
 - do_nothing, 3-65
 - do_something, 3-66
 - get_album, 3-66
 - get_event, 3-66
 - get_eventtype, 3-67
 - get_image, 3-67
 - get_info, 3-67
 - get_info_raw, 3-67
 - get_message, 3-67
 - get_name, 3-68
 - get_objclass, 3-68
 - get_objname, 3-68
 - get_oid, 3-68
 - get_platform, 3-69
 - get_raw_event, 3-66
 - get_time, 3-69
 - handled, 3-69
 - set_album, 3-69
 - set_event_raw, 3-70
 - set_eventtype, 3-70
 - set_image, 3-70
 - set_info_raw, 3-70
 - set_message, 3-70
 - set_name, 3-71
 - set_objclass, 3-71
 - set_objname, 3-71
 - set_oid, 3-71
 - set_time, 3-71
 - something_to_do, 3-72
- method types, 3-63
- operator overloading, 3-73

D

- DataUnit, 2-31, 2-32, 2-33, 2-42
- DataUnit class, 2-28, 2-59
 - constructors, 2-60 to 2-62
 - destructor, 2-62

- member functions
 - catenate, 2-65
 - chp, 2-65
 - cmp, 2-65
 - copy, 2-66
 - copyin, 2-66
 - copyout, 2-67
 - equiv, 2-67
 - fragment, 2-67
 - hash, 2-68
 - size, 2-68
 - unshare, 2-68
- memory management, 2-59
- operators, 2-62

defined types

- AsniInt, 3-22
- CCB, 3-22
- CDU, 3-22
- DU, 3-22
- FBits, 3-22
- DeleteReq class, 4-18
- DeleteRes class, 4-19
- design objectives, high-level PMI, 3-1
- Dictionary class, 2-69

member functions

- lookup, 2-70
- num_elems, 2-70
- position, 2-70
- table, 2-71

- dispatch_recursive function, 3-24
- Distinguished Name (DN), encoding of, 2-29
- DuplicateOI class, 4-20
- DupMessageId class, 4-21

E

- element naming, 8-14
- EMAgent class, 8-107
 - access member functions
 - get_administrative_state, 8-108
 - get_operational_state, 8-108
 - set_administrative_state, 8-109
- EMCmpAgent class, 8-12, 8-112
 - access member functions
 - add_managed_object, 8-117
 - get_agent_address_info, 8-118
 - get_agent_address_tag, 8-118
 - get_application_entity_invoke_id,

- 8-122
- get_application_entity_qualifier, 8-121
- get_application_entity_title, 8-120
- get_application_process_invoke_id, 8-122
- get_dn, 8-115
- get_managed_objects, 8-116
- get_mpa_address_info, 8-116
- get_name_translation, 8-121
- get_network_sap, 8-117
- get_presentation_selector, 8-119
- get_session_selector, 8-119
- get_transport_selector, 8-120
- remove_managed_object, 8-117
- set_agent_address_info, 8-118
- set_agent_address_tag, 8-119
- set_application_entity_invoke_id, 8-122
- set_application_entity_qualifier, 8-122
- set_application_entity_title, 8-121
- set_application_process_invoke_id, 8-123
- set_dn, 8-115
- set_managed_objects, 8-117
- set_mpa_address_info, 8-116
- set_name_translation, 8-121
- set_network_sap, 8-118
- set_presentation_selector, 8-119
- set_session_selector, 8-120
- set_transport_selector, 8-120
- EMCmipAgentDn class, 8-109
 - access member functions
 - system_name, 8-110
 - unique_name, 8-111
- EMdataset class, 1-16
- EMdynamicDataset class, 1-16
- EMgraph class, 1-18
- EMIntegerSet class, 8-13, 8-32
- EMObject class, 8-10, 8-47
 - member functions, 8-11
- EMRpcAgent class, 8-13, 8-126
 - access member functions
 - add_schema, 8-131
 - get_dn, 8-129
 - get_get_community_string, 8-130
 - get_schemas, 8-131
 - get_set_community_string, 8-130
 - remove_schema, 8-132
 - set_dn, 8-130
 - set_get_community_string, 8-130
 - set_schemas, 8-131
 - set_set_community_string, 8-131
- EMRpcAgent attributes, 8-127
- EMRpcAgentDn class, 8-123
 - access member functions
 - system_name, 8-125
 - unique_name, 8-125
- EMSnmpAgent class, 8-12, 8-135
 - access member functions
 - add_supported_mib, 8-143
 - get_access_control_enforcement, 8-144
 - get_access_control_mechanism, 8-145
 - get_dn, 8-140
 - get_get_community_string, 8-141
 - get_management_protocol, 8-143
 - get_set_community_string, 8-141
 - get_supported_mibs, 8-142
 - get_system_title, 8-140
 - get_transport_address, 8-142
 - remove_supported_mib, 8-143
 - set_access_control_enforcement, 8-144
 - set_access_control_mechanism, 8-145
 - set_dn, 8-140
 - set_get_community_string, 8-141
 - set_management_protocol, 8-144
 - set_set_community_string, 8-142
 - set_supported_mibs, 8-143
 - set_system_title, 8-141
 - set_transport_address, 8-142
- EMSnmpAgentDn class, 8-132
- EMStaticDataset class, 1-17
- EMStatus class, 8-13, 8-29
- EMTopoNode class, 8-12, 8-59
 - access methods
 - add_array_visible_child, 8-89
 - add_display_status, 8-74
 - add_link, 8-76
 - add_managed_object, 8-80
 - add_monitor_visible_child, 8-85
 - add_parent, 8-75
 - add_propagate_peer, 8-72
 - add_user_datum, 8-82
 - get_array_cell_width, 8-88
 - get_array_hidden_children, 8-90
 - get_array_num_columns, 8-87
 - get_array_orientation, 8-86

- get_array_visible_children, 8-88
- get_bus_logical_locations, 8-90
- get_children, 8-74
- get_children_containers_only, 8-74
- get_cmip_agents, 8-80
- get_display_status, 8-73
- get_display_statuses, 8-73
- get_dn, 8-68
- get_geographical_location, 8-78
- get_is_severity_propagated, 8-71
- get_layer_name, 8-79
- get_links, 8-76
- get_logical_location, 8-77
- get_logical_locations, 8-77
- get_managed_objects, 8-79
- get_monitor_hidden_children, 8-86
- get_monitor_max_visible_children, 8-86
- get_monitor_rotation, 8-84
- get_monitor_visible_children, 8-85
- get_name, 8-69
- get_parents, 8-75
- get_propagate_peers, 8-71
- get_propagated_severity, 8-71
- get_rpc_agents, 8-81
- get_severity, 8-70
- get_snmp_agents, 8-80
- get_state, 8-72
- get_topology_pathnames, 8-69
- get_type_name, 8-70
- get_user_data, 8-81
- get_user_datum, 8-82
- get_view_background_image_filename, 8-83
- get_view_default_geo_area, 8-84
- get_view_map_config_filename, 8-83
- remove_array_visible_child, 8-89
- remove_display_status, 8-74
- remove_link, 8-76
- remove_managed_object, 8-80
- remove_monitor_visible_child, 8-86
- remove_parent, 8-75
- remove_propagate_peer, 8-72
- remove_user_datum, 8-82
- set_array_cell_width, 8-88
- set_array_num_columns, 8-87
- set_array_orientation, 8-87
- set_array_visible_children, 8-89
- set_bus_logical_locations, 8-90
- set_display_statuses, 8-73
- set_dn, 8-69
- set_geographical_location, 8-78
- set_is_severity_propagated, 8-71
- set_layer_name, 8-79
- set_links, 8-76
- set_logical_location, 8-77
- set_logical_locations, 8-78
- set_managed_objects, 8-79
- set_monitor_rotation, 8-85
- set_monitor_visible_children, 8-85
- set_name, 8-69
- set_parents, 8-75
- set_propagate_peers, 8-72
- set_severity, 8-70
- set_state, 8-73
- set_type_name, 8-70
- set_user_data, 8-81
- set_view_background_image_filename, 8-83
- set_view_default_geo_area, 8-84
- set_view_map_config_filename, 8-83
- event subscription, 8-91
- EMTopoNodeDn class, 8-55
- EMTopoPlatform class, 8-7, 8-39
- methods
 - get_attributes_by_mo (), 8-8
 - set_attributes_by_mo (), 8-9
- EMTopoType class, 8-11, 8-96
- access member functions
 - add_legal_child, 8-101
 - add_user_data_attribute_name, 8-102
 - get_all_base_types, 8-100
 - get_base_type, 8-99
 - get_dn, 8-99
 - get_layer_name, 8-101
 - get_legal_children, 8-100
 - get_sub_types, 8-100
 - get_user_data_attribute_names, 8-101
 - remove_user_data_attribute_name, 8-102
 - set_base_type, 8-100
 - set_dn, 8-99
 - set_layer_name, 8-101
 - set_user_data_attribute_names, 8-102
- callback function, 8-105
- static member functions
 - event subscription, 8-105
 - is_array, 8-104
 - is_bus, 8-104
 - is_container, 8-103

- is_device, 8-104
- is_link, 8-104
- is_monitor, 8-103
- is_view, 8-103
- EMTopoTypeDn class, 8-93
- encoded data, 3-3
- Error class, 3-24, 3-72
 - Error types and strings, 3-76
 - member functions
 - error_to_string, 3-74
 - get_error_string, 3-74
 - get_error_type, 3-75
 - reset_error, 3-75
 - set_error, 3-75
 - set_error_entry_callback, 3-76
 - set_error_string, 3-75
 - set_error_type, 3-75
- error handling, 3-24
- ErrorResUnexp class, 4-22
- event dispatching, 3-24
 - functions, 3-25
- event handling
 - Viewer API, 1-9
- event registration, 8-105
- event report response callback, 9-57
- event sieves, 3-1
- EventReq class, 4-23

F

- filter, 3-6
- filtering derivation, 3-7

G

- GenInt class, 2-71
 - member functions
 - &operator %=, 2-76
 - &operator &=, 2-77
 - &operator *=, 2-75
 - &operator +=, 2-74
 - &operator /=, 2-76
 - &operator -=, 2-75
 - &operator ^=, 2-78
 - &operator |=, 2-78
 - bits, 2-72
 - div, 2-73

- encode, 2-73
- format, 2-73
- operator, 2-76, 2-77
- operator -, 2-74
- operator !, 2-75
- operator !=, 2-79
- operator %, 2-76
- operator &, 2-77
- operator *, 2-75
- operator +, 2-74
- operator /, 2-75
- operator ==, 2-78
- operator >, 2-76
- operator ^, 2-78
- operator |, 2-77
- operator ~, 2-78
- operator I32, 2-73
- operator U32, 2-74
- sign, 2-72
- size, 2-72
- get_error_string(), 3-74
- get_error_type(), 3-75
- get_sys_dn function, 9-58
- GetListErr class, 4-24
- getPackage Action
 - code examples
 - output, 3-11
- getPackagesByOC Action
 - code examples
 - input and output syntax, 3-15
 - output, 3-15
- GetReq class, 4-25
- GetRes class, 4-26
- grapher API, 1-15

H

- Hash Class, 2-79
- Hashdeclare macro, 2-81
- HashImpl class, 2-81
 - member functions, 2-83
- Hdict class, 2-85
 - member functions
 - lookup, 2-86
 - num_elems, 2-86
 - position, 2-87
 - set, 2-87
 - table, 2-87

high level usage, 3-2
Hrefdict class, 2-87

- member functions
 - lookup, 2-89
 - num_elems, 2-89
 - position, 2-89
 - set, 2-90
 - table, 2-89

I

Image class, 3-4, 3-77

- constructors
 - copy, 3-80
 - default, 3-80
 - general, 3-80
- events, 3-115
- global functions, related
 - fdn2formal, 3-115
 - fdn2oi, 3-115
 - name2oc, 3-116
 - oc2name, 3-116
 - oi2fdn, 3-116
- member functions
 - attr_changed, 3-81
 - attr_exists, 3-82
 - boot, 3-83
 - call, 3-84
 - create, 3-85
 - create_within, 3-85
 - destroy, 3-86
 - exists, 3-87
 - find_by_nickname, 3-88
 - find_by_objname, 3-89
 - find_by_oi, 3-88
 - first_album, 3-89
 - get, 3-89
 - get_attr_last_error, 3-93
 - get_attr_names, 3-90
 - get_attr_numerrors, 3-93
 - get_attr_prop, 3-91
 - get_attr_trackmode, 3-93
 - get_dbl, 3-94
 - get_encoded_oi, 3-97
 - get_long, 3-95
 - get_nickname, 3-95
 - get_objclass, 3-96
 - get_objname, 3-96

- get_oc, 3-97
- get_oi, 3-97
- get_param_syntax, 3-97
- get_prop, 3-98
- get_raw, 3-99
- get_result_syntax, 3-100
- get_set, 3-100
- get_set_dbl, 3-100
- get_set_gint, 3-101
- get_set_long, 3-101
- get_set_raw, 3-102
- get_set_str, 3-102
- get_str, 3-103
- get_userdata, 3-104
- get_when_syntax, 3-104
- get-gint, 3-94
- is_in_album, 3-104
- revert, 3-105
- send_event, 3-106
- set, 3-107
- set_attr_prop, 3-107
- set_dbl, 3-107
- set_from_ref, 3-108
- set_gint, 3-108
- set_long, 3-109
- set_nickname, 3-109
- set_objclass, 3-109
- set_prop, 3-110
- set_raw, 3-110
- set_str, 3-110
- set_userdata, 3-111
- shutdown, 3-112
- start, 3-112
- start_boot, 3-112
- start_create, 3-112
- start_create_within, 3-113
- start_destroy, 3-113
- start_raw, 3-113
- start_shutdown, 3-113
- start_store, 3-113
- store, 3-114
- U32 num_albums, 3-105
- when, 3-114

method types, 3-78

operators, 3-81

properties, 3-4, 3-92, 3-98

InexpError class, 4-75

InvalidActionArg class, 4-27

InvalidAttrVal class, 4-28

InvalidEventArgs class, 4-29
InvalidFilter class, 4-30
InvalidOI class, 4-31
InvalidOperation class, 4-32
InvalidOperator class, 4-33
InvalidScope class, 4-34

K

KernelMessageSAP class, 4-39

L

LinkedResUnexp class, 4-35
locational flexibility and transparency, 3-1
low level primitives, 3-2
LPP (Lightweight Presentation Protocol), 4-1

M

makefile, 8-15
managed object, creation rules, 9-33
manipulating objects, 3-2
MDR (Meta Data Repository), 3-7

actions

- getAllDocuments
- getAsn1Module
- getAttribute
- getDocument
- getObjectClass
- getOidName

actions sample program, 3-17

description, 3-7

symbolic constants, 3-18

Message class, 4-36

messages

viewer to application, 1-10

MessageSAP class, 4-39

member functions

- cancel_callback, 4-41
- new_id, 4-42
- receive_request, 4-42
- send, 4-43

MessQOS Class, 4-45

MessScope Class, 4-45

MIS independence, 3-1

MIS-MIS awareness, 8-15

MissingAttrVal class, 4-47

MistypedArg class, 4-48

MistypedError class, 4-49

MistypedOp class, 4-50

MistypedRes class, 4-51

Morf class, 3-3, 3-116

constructors, 3-118 to 3-120

copy constructor, 3-119

virtual constructor, 3-119

destructor, 3-120

member functions

extract, 3-121

get, 3-123

get, code example, 3-124, 3-127

get_dbl, 3-125

get_gint, 3-126

get_long, 3-126

get_member_names, 3-126

get_memname, 3-126

get_platform, 3-127

get_syntax, 3-129

get_type, 3-130

get_type(), code example, 3-130

get_str, 3-129

has_value, 3-130

is_any, 3-131

is_choice, 3-131

is_list, 3-131

is_sequence, 3-131

is_set, 3-132

num_elements, 3-132

ref, 3-132

set, 3-132

set_any, 3-133

set_dbl, 3-133

set_gint, 3-133

set_long, 3-133

set_memname, 3-134

set_str, 3-134

set_value, 3-134

split_array, 3-135

split_queue, 3-135

operators

assignment operator, 3-120

cast operator, 3-121

comparison operator, 3-121

Morf method types, 3-117

Morf::get_type(), code example, 3-124

MorfBuilder class, 3-135

member functions

get_error_string, 3-146

get_error_type, 3-145

get_prop, 3-144

get_raw, 3-138

select_choice, 3-140

set, 3-142

set_prop, 3-147

set_raw, 3-141

set_syntax, 3-141

validate, 3-143

MRM (Message Routing Module), 4-1

N

NCAsyncResIterator class, 7-5

NCI (Nerve Center Interface), 7-1

NC requests, 7-3

asynchronous launches, 7-4

synchronous launches, 7-3

NCI functions, 7-14

nci_action_add, 7-14

nci_action_delete, 7-14

nci_async_request_start, 7-15

nci_condition_add, 7-16

nci_condition_delete, 7-16

nci_condition_get, 7-16

nci_init, 7-17

nci_parse_handle, 7-18

nci_pollrate_add, 7-18

nci_pollrate_delete, 7-19

nci_request_delete, 7-19

nci_request_dump, 7-19

nci_request_info, 7-20

nci_request_list, 7-20

nci_request_start, 7-21

alternative syntax, 7-21 to 7-23

nci_severity_add, 7-23

nci_severity_delete, 7-24

nci_state_add, 7-24

nci_state_delete, 7-24

nci_state_get, 7-25

nci_template_add, 7-25

nci_template_copy, 7-26

nci_template_create, 7-26

nci_template_delete, 7-26

nci_template_find, 7-27

nci_template_revert, 7-27

nci_template_store, 7-28

nci_transition_add, 7-28

nci_transition_delete, 7-29

nci_transition_find, 7-29

nci_transition_get, 7-30

NCI global variables, 7-13

nci_error_reason, 7-13

topoNodeId argument, 7-13

NCI library

functions, 7-11

initialization routines, 7-17

NCI library classes, list of, 7-5

request templates, 7-1 to 7-30

sample program, 7-30 to 7-33

NCParsedReqHandle class, 7-7

NCTopoInfoList class, 7-9

Nerve Center request template, 7-1

NoSuchAction class, 4-52

NoSuchActionArg class, 4-53

NoSuchAttr class, 4-54

NoSuchEvent class, 4-55

NoSuchEventArg class, 4-56

NoSuchMessageId class, 4-57

NoSuchOC class, 4-58

NoSuchOI class, 4-59

NoSuchRefOI class, 4-60

notifications, clients, 8-105

O

object

absolute name, 3-2

nickname, 3-2

relationships, 3-3

directed nature, 3-3

set membership, 3-3

object services API enhancements

create request service, 9-33

delete request service, 9-42

get request service, 9-8

set request service, 9-19

objects

manipulating, 3-2

naming, 3-2, 3-5

ObjReqMess class, 4-61

ObjResMess class, 4-62

ODT (object development tools) *See* OS API, 9-1

OID (Object Identifier), 2-37

Oid class, 2-90

- constructors, 2-91

- member functions

 - add_in, 2-92

 - add_last_in, 2-92

 - append, 2-92

 - copy_oid, 2-93

 - format, 2-93

 - get_id, 2-93

 - is_same_prefix, 2-93

 - num_ids, 2-94

 - print, 2-68, 2-94

- object identifier, 2-90

- operators, 2-91

OpCancelled class, 4-63

OS API (object services API), 9-1

- action request service, 9-26

- action response example, 9-31

- create request service, 9-33

- create response callback, 9-39

- debugging flags, 9-58

- delete request service, 9-42

- delete response callback, 9-49

- event report request service, 9-53

- get request service, 9-8

- get response callback, 9-15

- get_graphstr_rdn functions, 9-59

- get_sys_dn function, 9-58

- interface descriptions and examples, 9-7

- operational flow, 9-2

- send_event_req example, 9-54

- service request function parameters, 9-3

- service response callback function parameters, 9-7

- set request service, 9-19

- set response callback, 9-24

- supporting functions for example code, 9-57

P

PasswordTty class, 3-148

Platform class, 3-149

- constructors, 3-151

- DEFAULT_TIMEOUT constant, 3-18

- events, 3-167

- GETENV macro, 3-168

- member functions

 - cleanup_def_platform, 3-152

 - connect, 3-155

 - default_platform, 3-155

 - disconnect, 3-156

 - find_album_by_nickname, 3-159

 - find_image_by_nickname, 3-159

 - find_image_by_objname, 3-159

 - find_image_by_oid, 3-159

 - get_attr_coder, 3-160

 - get_authorized_applications, 3-160

 - get_authorized_features, 3-161

 - get_connection, 3-162

 - get_fdn, 3-162

 - get_fullname, 3-163

 - get_plat_id, 3-163

 - get_prop, 3-163

 - get_raw_sap, 3-165

 - get_shortcode, 3-165

 - get_when_syntax, 3-165

 - replace_discriminator, 3-164

 - set_attr_coder, 3-166

 - set_default_platform, 3-166

 - set_prop, 3-166

 - start_connect, 3-166

 - start_disconnect, 3-167

 - when, 3-167

- operators, 3-151

- properties, 3-4, 3-163

- TIME_OUT property, 3-18

platform method types, 3-150

PMI (Portable Management Interface), 3-1, 4-1

POC (persistent object classes), 8-10

primary CMIS message types, 4-4

ProcessFailure class, 4-64

Q

Queue class, 2-97

Queuedeclare macro, 2-100

R

RCL (Request Condition Language), 7-16

ReqMess class, 4-65

reset_error(), 3-75

ResMess class, 4-66

ResourceLimit class, 4-67

S

- SAPs (Service Access Points)
 - transport-Dependent, 4-1
 - transport-Independent, 4-1
- ScopedReqMess class, 4-68
- scoping, 3-3
- scoping parameters, 3-6
- send_event_req example, 9-54
- send_get_req() function, 9-8
- send_set_req() function, 9-19
- service request function parameters, 9-3
- set request service, 9-19
- set_error(), 3-75
- set_error_string(), 3-75
- set_error_type(), 3-75
- set_indication_handler(), 1-10
- SetListErr class, 4-69
- SetReq class, 4-70
- SetRes class, 4-71
- string constants, 3-19
- SyncNotSupp class, 4-72
- Syntax class, 3-168
 - member functions
 - expansion, 3-171
 - get, 3-171
 - get_coder, 3-171
 - get_member_names, 3-171
 - get_members, 3-172
 - get_memname, 3-172
 - get_platform, 3-172
 - get_raw, 3-172
 - get_type, 3-173
 - is_any, 3-173
 - is_choice, 3-173
 - is_list, 3-174
 - is_sequence, 3-174
 - is_set, 3-174
 - member, 3-174
 - set_coder, 3-175
- Syntax method types, 3-169
- Syntax::get_type(), code example, 3-173

T

- TimedOut class, 4-73
- Timer class, 2-101
 - global functions, related
 - getGeneralizedTime, 2-103

- post_timer, 2-102
- post_timer_handler, 2-102
- purge_timer, 2-103
- purge_timer_data, 2-103
- purge_timer_handler, 2-103

- topology API, 8-1
 - class overview, 8-5
 - class reference, 8-29
 - concepts, 8-14
 - examples, 8-15 to 8-25
 - GDMO relationship, 8-5
 - general description, 8-3
 - relationship to PMI, 8-6
- topology node names
 - duplicates, 8-14

U

- UnexpChildOp class, 4-74
- UnexpRes class, 4-76
- UnrecError class, 4-77
- UnrecLinkId class, 4-78
- UnrecMessageId class, 4-79
- UnrecOp class, 4-80
 - variables
 - MessagePtr, 4-82
 - MessBaseType, 4-83
 - MessId, 4-81
 - MessMode, 4-82
 - MessScopeType, 4-82
 - MessSync enumerator, 4-83
 - MessType enumerator, 4-84
 - MESSTYPE_MAX, 4-86
 - ResponseHandle, 4-86
 - SendResult, 4-86
- utility classes, 8-13

V

- Viewer API
 - communication protocol, 1-6
 - overview, 1-2
- viewer event messages, 1-10
- viewerAPI class, 1-2
 - actions, 1-6
 - viewerPopupMessage dialog, 1-8
 - viewerPopupQuestion dialog, 1-8

member functions

- set_indication_handler, 1-5
- viewerapi_build_target, 1-4
- viewerapi_send_request, 1-3, 1-4
- viewerapi_send_request_unconfirmed,
1-4, 1-5
- viewerapi_start_send_request, 1-3, 1-5

sample programs, 1-15

viewer event handling, 1-9

- ViewRegisterEvents, 1-11

viewer event messages

- duLayerChangeEvent, 1-13
- duObjectCreationEvent, 1-14
- duObjectDeletionEvent, 1-14
- duObjectDeselectedEvent, 1-12
- duObjectSelectedEvent, 1-11
- duPopupMenuEvent, 1-12
- duRegisterForEvents, 1-15
- duToolsMenuEvent, 1-13
- duViewChangeEvent, 1-14

W

Waiter class, 3-175

constructors, 3-176

member functions

- cancel, 3-179
- clobber, 3-180
- complete, 3-180
- dec, 3-180
- get_current_event, 3-181
- get_data, 3-181
- get_except, 3-181
- inc, 3-181
- num_clobbered, 3-182
- ref, 3-182
- send_resp, 3-182
- time_remaining, 3-183
- wait, 3-183
- waitmore, 3-183
- was_completed, 3-183
- when_canceled, 3-184
- when_done, 3-184
- when_resp, 3-184
- when_tick, 3-186

method types, 3-175

operators, 3-179

